Reachability Analysis over Term Rewriting Systems

Guillaume Feuillade, Thomas Genet, Valérie Viet Triem Tong

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Reachability Analysis over Term Rewriting Systems

Guillaume Feuillade , Thomas Genet , Valérie Viet Triem Tong

N°4970

23 Octobre 2003

THÈMES 1 et 2

ISSN 0249-6399 ISRN INRIA/RR--4970--FR+ENG

Reachability Analysis over Term Rewriting Systems

Guillaume Feuillade , Thomas Genet , Valérie Viet Triem Tong

Thèmes 1 et 2 — Réseaux et systèmes — Génie logiciel
et calcul symbolique
Projet S4 et Lande

Rapport de recherche n ° 4970 — 23 Octobre 2003 — 40 pages

Abstract: This paper surveys some techniques and tools for achieving reachability analysis over term rewriting systems. The core of those techniques is a generic tree automata completion algorithm used to compute in an exact or approximated way the set of descendants (or reachable terms). This algorithm has been implemented in the Timbuk tool. Furthermore, we show that classes with regular sets of descendants of the literature corresponds to specific instances of the tree automata completion algorithm and can thus be efficiently computed by Timbuk. An extension of the completion algorithm to conditional term rewriting systems and some applications are also presented.

Key-words: Term Rewriting Systems, Reachability, Tree Automata

(Résumé : tsvp)

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

Analyse d'atteignabilité sur les systèmes de réécriture

Résumé : Cet article regroupe un ensemble de techniques destinées à l'analyse d'atteignabilité sur les systèmes de réécriture. La technique principale est un algorithme générique de complétion des automates d'arbre qui permet de calculer de façon exacte ou approchée l'ensemble des descendants (ou termes accessibles). Cet algorithme est implanté dans l'outil Timbuk. Nous montrons également que les classes ayant des ensembles de descendants réguliers, que l'on trouve dans la littérature, correspondent à des instances spécifiques de l'algorithme de complétion d'automates. En conséquence, ces classes régulières peuvent être calculées de façon efficace par Timbuk. Nous proposons également une extension de l'algorithme de complétion au cas des systèmes de réécriture conditionnels ainsi que des applications.

Mots-clé : Systèmes de réécriture, Atteignabilité, Automates d'arbres

Introduction

Given a term rewriting system $\mathcal{R}$ and two ground terms s and t, we focus on proving automatically that $s \rightarrow_{\mathcal{R}}^* t$ or $s \not\rightarrow_{\mathcal{R}}^* t$. This problem has several applications in equational proofs used in theorem proving or in proof assistants as well as in verification where term rewriting systems can be used to model programs. The reachability problem is known to be decidable for Term Rewriting Systems (TRS for short) which are terminating. However, in automated deduction and in verification, systems considered in practice are rarely terminating and, even when they are, automatically proving their termination is difficult. On the other hand, reachability is known to be decidable on several syntactic classes of term rewriting systems (not necessarily terminating nor confluent). On those classes, the technique used to prove reachability is rather different and is based on the computation of the set $\mathcal{R}^\star(E)$ of $\mathcal{R}$-descendants (or $\mathcal{R}$-reachable terms) of an initial set of terms E. For those classes, $\mathcal{R}^\star(E)$ is a regular tree language and can thus be represented using a *tree automaton*. Tree automata offer a finite way to represent infinite (regular) sets of reachable terms when a non terminating term rewriting system is under concern.

In this paper, our aim is to propose a common, simple, efficient and implemented algorithm for computing known decidable regular classes as well as to construct some approximation when it is not decidable. This algorithm is essentially a *completion* of a *tree automata*, thus taking advantage of an algorithm similar to the Knuth-Bendix *completion* [20] in order not to restrict to a specific syntactic class of term rewriting systems and *tree automata* in order to deal efficiently with infinite sets of reachable terms produced by non-terminating term rewriting systems.

This algorithm is implemented in the Timbuk tool [14]. However, as we will see in the following, our implementation does not cover *every* decidable class since this would have led to an inefficient tool. As an example, for dealing with non left-linear TRSs, one can refine the algorithm we propose by applying determinisation after each step of tree automata completion. In this way, one may obtain a more general theorem covering the non left-linear case without restriction. However, since determinisation is an exponential-time operation, this would not be realistic in practice. Thus, we choose to stick to the basic completion algorithm and give some conditions sufficient for covering the non left-linear case in many practical cases.

The paper is organized as follows. In section 1 we recall the basic notations for term rewriting systems and tree automata. Then, in section 2 we recall the known regular classes for descendants. Section 3 presents the tree automata completion algorithm and the result for over-approximation of $\mathcal{R}^\star(E)$ for any TRS $\mathcal{R}$ and any initial regular language E. In section 4 we give some sufficient conditions for the tree automata completion to compute exactly $\mathcal{R}^\star(E)$ for any TRS $\mathcal{R}$ and any initial regular language E. In this section we also show how regular classes of the literature can be obtained using tree automata completion. Then in section 5 we give some formal and practical tools for guiding the approximation construction when $\mathcal{R}^\star(E)$ is not regular. In section 6, some applications of $\mathcal{R}^\star(E)$ are presented: sufficient completeness, strong non-termination proof and reachability testing.

In section 7, we present the algorithmic optimisation of matching in automata in Timbuk and some indications on its efficiency. Finally, an extension of the tree automata completion algorithm for conditional term rewriting systems is given in section 8.

1 Formal background

Comprehensive surveys can be found in [8, 1] for term rewriting systems, in [5, 17] for tree automata and tree language theory.

Let $\mathcal{F}$ be a finite set of symbols, each associated with an arity function ar, and let $\mathcal{X}$ be a countable set of variables. $\mathcal{T}(\mathcal{F},\mathcal{X})$ denotes the set of terms, and $\mathcal{T}(\mathcal{F})$ denotes the set of ground terms (terms without variables). The set of variables of a term t is denoted by $\mathcal{V}ar(t)$. The domain and range of a mapping will be denoted respectively by $\mathcal{D}om$ and $\mathcal{R}an$. A substitution is a mapping σ from $\mathcal{X}$ into $\mathcal{T}(\mathcal{F},\mathcal{X})$, which can uniquely be extended to an endomorphism of $\mathcal{T}(\mathcal{F},\mathcal{X})$. Its domain $\mathcal{D}om(\sigma)$ is $\{x \in X \mid x\sigma \neq x\}$. A position p for a term t is a word over $\mathbb{N}$. The empty sequence ϵ denotes the top-most position. The set $\mathcal{P}os(t)$ of positions of a term t is inductively defined by:

- $\mathcal{P}os(t) = \{\epsilon\}$ if $t \in \mathcal{X}$
- $\mathcal{P}os(f(t_1,\ldots,t_n)) = \{\epsilon\} \cup \{i.p \mid 1 \leq i \leq n \text{ and } p \in \mathcal{P}os(t_i)\}$

If $p \in \mathcal{P}os(t)$, then $t|_p$ denotes the subterm of t at position p and $t[s]_p$ denotes the term obtained by replacement of the subterm $t|_p$ at position p by the term s. For any term $s \in \mathcal{T}(\mathcal{F},\mathcal{X})$, we denote by $\mathcal{P}os_{\mathcal{F}}(s)$ the set of functional positions in s, i.e. $\{p \in \mathcal{P}os(s) \mid p \neq \epsilon \text{ and } \mathcal{R}oot(s|_p) \in \mathcal{F}\}$ where $\mathcal{R}oot(t)$ denotes the symbol at position ϵ in t. Conversely, we denote by $\mathcal{P}os_{\mathcal{X}}(s)$ the set of variable positions in s, i.e. $\mathcal{P}os_{\mathcal{X}}(s) = \mathcal{P}os(s) \setminus \mathcal{P}os_{\mathcal{F}}(s)$.

A term rewriting system $\mathcal{R}$ is a set of *rewrite rules* $l \to r$, where $l, r \in \mathcal{T}(\mathcal{F},\mathcal{X})$, $l \notin \mathcal{X}$, and $\mathcal{V}ar(l) \supseteq \mathcal{V}ar(r)$. A rewrite rule $l \to r$ is *left-linear* (resp. *right-linear*) if each variable of l (resp. r) occurs only once. A rule is linear if it is both left and right-linear. A TRS $\mathcal{R}$ is linear (resp. left-linear, right-linear) if every rewrite rule $l \to r$ of $\mathcal{R}$ is linear (resp. left-linear, right-linear). The TRS $\mathcal{R}$ induce a rewriting relation $\to_{\mathcal{R}}$ on terms whose reflexive transitive closure is denoted by $\to^{\star}_{\mathcal{R}}$. The set of $\mathcal{R}$-descendants of a set of ground terms E is $\mathcal{R}^{\star}(E) = \{t \in \mathcal{T}(\mathcal{F}) \mid \exists s \in E \text{ s.t. } s \to^{\star}_{\mathcal{R}} t\}$. We extend this notation to terms in the following way: $\mathcal{R}^{\star}(s) = \mathcal{R}^{\star}(\{s\})$. We denote by $IRR(\mathcal{R})$ the set of terms irreducible by $\mathcal{R}$ and by $\mathcal{R}^{!}(E)$ the set of $\mathcal{R}$-normal forms of E, i.e. $\mathcal{R}^{!}(E) = \mathcal{R}^{\star}(E) \cap IRR(\mathcal{R})$.

Let $\mathcal{Q}$ be a finite set of symbols, with arity 0, called *states*. $\mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ is called the set of *configurations*. A *transition* is a rewrite rule $c \to q$, where $c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $q \in \mathcal{Q}$. A *normalized transition* is a transition $c \to q$ where $c = q' \in \mathcal{Q}$ or $c = f(q_1,\ldots,q_n)$, $f \in \mathcal{F}$, $ar(f) = n$, and $q_1,\ldots,q_n \in \mathcal{Q}$. A bottom-up non-deterministic finite tree automaton (tree automaton for short) is a quadruple $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, where $\mathcal{Q}_f \subseteq \mathcal{Q}$ and Δ is a set of normalized transitions. A tree automaton is *deterministic* if there are no two rules with the same left-hand side. The rewriting relation induced by the transitions of $\mathcal{A}$ (the set Δ) is denoted by $\to_{\mathcal{A}}$. The tree language recognized by a state q in $\mathcal{A}$ is

$\mathcal{L}(\mathcal{A}, q) = \{t \in \mathcal{T}(\mathcal{F}) \mid t \to_{\mathcal{A}}^{\star} q\}$. The language recognized by $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) = \bigcup_{q \in \mathcal{Q}_f} \mathcal{L}(\mathcal{A}, q)$. A tree language is regular if and only if it can be recognized by a tree automaton. By notation abuse, we will often note $q \in \mathcal{A}$ and $t \to q \in \mathcal{A}$ respectively for $q \in \mathcal{Q}$ and $t \to q \in \Delta$.

2 Existing solutions

The basic reachability problem we are going to consider is the following: given a term rewriting system $\mathcal{R}$ and two terms s and t can we decide whether $s \to_{\mathcal{R}}^{\star} t$ or not? In this part, we focus on the existing solutions designed for particular cases.

The simplest case is when $\mathcal{R}$ is terminating: to decide whether $s \to_{\mathcal{R}}{}^{\star} t$ or not it is enough to see if $t \in \mathcal{R}^{\star}(s)$ since $\mathcal{R}^{\star}(s)$ is finite and computable.

When $\mathcal{R}$ is not terminating, deciding reachability needs some additional formal tools, namely tree automata, in order to finitely represent the infinite set $\mathcal{R}^{\star}(s)$ and then check if $t \in \mathcal{R}^{\star}(s)$. Many works are devoted to the construction of $\mathcal{R}^{\star}(E)$ for a regular language E and a term rewriting system $\mathcal{R}$ fulfilling some restrictions:

- $\mathcal{R}$ is either a ground TRS [7, 3].
- a right-linear and monadic TRS [28], i.e. right-hand sides of the rules are either variables or terms of the form $f(x_1, \ldots, x_n)$ where $f \in \mathcal{F}$ and $x_1, \ldots, x_n$ are variables.
- a linear and semi-monadic TRS [6], i.e. rules are linear and their right-hand sides are of the form $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$ and $\forall i = 1, \ldots, n$, t_i is either a variable or a ground term.
- a "decreasing" TRS [18], where "decreasing" means that every right-hand side is either a variable, or a term $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$, $ar(f) = n$, and $\forall i = 1, \ldots, n$, t_i is a variable, a ground term, or a term whose variables do not occur in the left-hand side.

On the other hand, for a given regular language E, $\mathcal{R}^{\star}(E)$ is not necessarily regular, even if $\mathcal{R}$ is a confluent and terminating linear TRS [17]. If $\mathcal{R}$ is not "decreasing", then $\mathcal{R}^{\star}(E)$ is not necessarily regular [18].

Another regular class was found by P. Réty [27] where restrictions are weaker on the TRS and stronger on the regular language E. The alphabet $\mathcal{F}$ is separated into a set of *defined symbols* $\mathcal{D} = \{f \mid \exists l \to r \in \mathcal{R} \text{ s.t. } \mathcal{R}oot(l) = f\}$ and constructor symbols $\mathcal{C} = \mathcal{F} \setminus \mathcal{D}$. The restriction on E is the following: E is the set of ground constructor instances of a linear term t, i.e. $E = \{t\sigma\}$ where $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is linear and $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{C})$. The restrictions on $\mathcal{R}$ are the following: for each rule $l \to r$

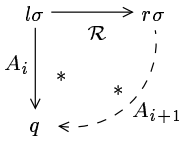
1. r is linear
2. for each position $p \in \mathcal{P}os_{\mathcal{F}}(r)$ such that $r|_p = f(t_1, \ldots, t_n)$ and $f \in \mathcal{D}$ we have that for all $i = 1 \ldots n$, t_i is a variable or a ground term
3. there is no nested function symbols in r

3 Tree Automata completion

In [11], we proposed a *tree automaton completion* algorithm for over-approximating $\mathcal{R}^\star(E)$ for left-linear term rewriting systems and a regular language E. The completion is parametrized by an abstraction function α mapping terms to states of the automaton.

Let us first recall the tree automata completion algorithm. Starting from a tree automaton $\mathcal{A}_0 = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta_0 \rangle$ and a left-linear[1] TRS $\mathcal{R}$, the aim of the approximation algorithm is to compute a tree automaton $\mathcal{A}'$ such that $\mathcal{L}(\mathcal{A}') \supseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A}_0))$. Approximations are used to show that terms recognized by a tree automaton $\mathcal{A}_{bad}$ are not reachable by rewriting terms of $\mathcal{L}(\mathcal{A}_0)$ with $\mathcal{R}$, i.e. $\forall s \in \mathcal{L}(\mathcal{A}_0)\ \forall t \in \mathcal{L}(\mathcal{A}_{bad}) : s \not\to_{\mathcal{R}}{}^* t$. For this, it is enough to show that $\mathcal{L}(\mathcal{A}') \cap \mathcal{L}(\mathcal{A}_{bad}) = \emptyset$ i.e., compute the automaton recognizing the intersection and show that the recognized language is empty.

The technique consists in successively computing tree automata $\mathcal{A}_1$, $\mathcal{A}_2$, ... such that $\forall i \geq 0 : \mathcal{L}(\mathcal{A}_i) \subseteq \mathcal{L}(\mathcal{A}_{i+1})$ and if $s \in \mathcal{L}(\mathcal{A}_i)$, such that $s \to_{\mathcal{R}} t$ then $t \in \mathcal{L}(\mathcal{A}_{i+1})$, until we get an automaton $\mathcal{A}_k$ with $k \in \mathbb{N}$ such that $\mathcal{L}(\mathcal{A}_k) = \mathcal{L}(\mathcal{A}_{k+1})$. Thus, $\mathcal{A}_k$ is a fixpoint and $\mathcal{A}_k$ also verifies $\mathcal{L}(\mathcal{A}_k) \supseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A}_0))$. More precisely, to construct $\mathcal{A}_{i+1}$ from $\mathcal{A}_i$, we achieve a *completion step* which consists in finding *critical pairs* between $\to_{\mathcal{R}}$ and $\to_{\mathcal{A}_i}$. For a substitution $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and a rule $l \to r \in \mathcal{R}$, a critical pair is an instance $l\sigma$ of l such that there exists $q \in \mathcal{Q}$ satisfying $l\sigma \to^\star_{\mathcal{A}_i} q$ and $r\sigma \not\to^\star_{\mathcal{A}_i} q$. For $r\sigma$ to be recognized as the same state and thus model the rewriting of $l\sigma$ into $r\sigma$, it is enough to join the critical pair:



and add the new transition $r\sigma \to q$ to $\mathcal{A}_{i+1}$. However, the transition $r\sigma \to q$ is not necessarily of the form $f(q_1, \ldots, q_n) \to q'$ and so has to be normalized first. For example, to normalize a transition of the form $f(g(a), h(q')) \to q$, we need to find some states q_1, q_2, q_3 and replace the previous transition by a set of normalized transitions: $\{a \to q_1, g(q_1) \to q_2, h(q') \to q_3, f(q_2, q_3) \to q\}$.

Assume that q_1, q_2, q_3 are new states, then adding the transition itself or its normalized form does not make any difference. Now, assume that $q_1 = q_2$, the normalized form becomes $\{a \to q_1, g(q_1) \to q_1, h(q') \to q_3, f(q_1, q_3) \to q\}$. This set of normalized transitions represents the regular set of non normalized transitions of the form $f(g^\star(a), h(q')) \to q$ which contains the transition we wanted to add initially but also many others. Hence, this is an approximation. We could have made an even more drastic approximation by identifying q_1, q_2, q_3 with q, for instance.

For every transition, there exists an equivalent set of normalized transitions. Normalization consists in decomposing a transition $s \to q$, into a set $Norm(s \to q)$ of normalized

[1] This restriction will be weakened in the following.

transitions. The method consists in abstracting subterms s' of s s.t. $s' \notin \mathcal{Q}$ by states of $\mathcal{Q}$. We first define the abstraction function as follows:

Definition 1 *(Abstraction function) Let $\mathcal{F}$ be a set of symbols, and $\mathcal{Q}$ a set of states. An* abstraction *function α maps every normalized configuration to a state:*

$$\alpha : \{f(q_1, \ldots, q_n) \mid f \in \mathcal{F}, ar(f) = n \text{ and } q_1, \ldots q_n \in \mathcal{Q}\} \mapsto \mathcal{Q}$$

In the following, for completed automata, we assume that $\mathcal{Q}$ is a set of states containing states of transitions Δ as well as states of the range of α. In particular, if the range of α is infinite (for instance, if completion does not terminate) then so is $\mathcal{Q}$.

Definition 2 *(Abstraction state) Let $\mathcal{F}$ be a set of symbols, and $\mathcal{Q}$ a set of states. For a given abstraction function α and for all configuration $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ the abstraction state of t, denoted by $top_\alpha(t)$, is defined by:*

1. *if $t \in \mathcal{Q}$, then $top_\alpha(t) = t$,*
2. *if $t = f(t_1, \ldots, t_n)$ then $top_\alpha(t) = \alpha(f(top_\alpha(t_1), \ldots, top_\alpha(t_n)))$.*

Definition 3 *(Normalization function) Let $\mathcal{F}$ be a set of symbols, $\mathcal{Q}$ a set of states, $s \to q$ a transition s.t. $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ and $q \in \mathcal{Q}$, and α an abstraction function. The set $Norm_\alpha(s \to q)$ of* normalized transitions *is inductively defined by:*

1. *if $s = q$, then $Norm_\alpha(s \to q) = \emptyset$, and*
2. *if $s \in \mathcal{Q}$ and $s \neq q$, then $Norm_\alpha(s \to q) = \{s \to q\}$, and*
3. *if $s = f(t_1, \ldots, t_n)$, then $Norm_\alpha(s \to q) =$*
   $\{f(top_\alpha(t_1), \ldots, top_\alpha(t_n)) \to q\} \cup \bigcup_{i=1}^{n} Norm_\alpha(t_i \to top_\alpha(t_i))$.

Example 1 *Let $\mathcal{F} = \{f, g, a\}$ and $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, where $\mathcal{Q} = \{q_0, q_1, q_2, q_3, q_4\}$, $\mathcal{Q}_f = \{q_0\}$, and $\Delta = \{f(q_1) \to q_0, g(q_1, q_1) \to q_1, a \to q_1\}$.*

- *The languages recognized by q_1 and q_0 are the following: $\mathcal{L}(\mathcal{A}, q_1)$ is the set of terms built on $\{g, a\}$, i.e. $\mathcal{L}(\mathcal{A}, q_1) = \mathcal{T}(\{g, a\})$, and $\mathcal{L}(\mathcal{A}, q_0) = \mathcal{L}(\mathcal{A}) = \{f(x) \mid x \in \mathcal{L}(\mathcal{A}, q_1)\}$.*
- *Let $s = f(g(q_1, f(a)))$, and α_1 be the abstraction function $\{a \mapsto q_4, f(q_4) \mapsto q_3, g(q_1, q_3) \mapsto q_2\}$. The normalization of transition $f(g(q_1, f(a))) \to q_0$ with abstraction α_1 is the following: $Norm_{\alpha_1}(f(g(q_1, f(a))) \to q_0) = \{f(q_2) \to q_0, g(q_1, q_3) \to q_2, f(q_4) \to q_3, a \to q_4\}$.*

Definition 4 *A* regular language substitution *(or a $\mathcal{Q}$-substitution) over an automaton $\mathcal{A}$ with a set of states $\mathcal{Q}$ is an application $\sigma : \mathcal{X} \mapsto \mathcal{Q}$. We can extend this definition to a morphism $\sigma : \mathcal{T}(\mathcal{F}, \mathcal{X}) \mapsto \mathcal{T}(\mathcal{F}, \mathcal{Q})$. We denote by $\Sigma(\mathcal{Q}, \mathcal{X})$ the set of regular language substitutions built over $\mathcal{Q}$ and $\mathcal{X}$.*

Definition 5 *(One step of automaton completion) Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton, $\mathcal{R}$ a TRS and α an abstraction function. The one step completed automaton $\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})$ is a tree automaton $\langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta' \rangle$ such that:*

$$\Delta' = \Delta \cup \bigcup_{l \to r \in \mathcal{R},\ q \in \mathcal{Q},\ \sigma \in \Sigma(\mathcal{Q},\mathcal{X}),\ l\sigma \to^\star_\Delta q} Norm_\alpha(r\sigma \to q)$$

Definition 6 *(Automaton completion) Let $\mathcal{A}$ be a tree automaton, $\mathcal{R}$ a TRS and α an abstraction function.*

- $\mathcal{A}^0_{\alpha,\mathcal{R}} = \mathcal{A}$
- $\mathcal{A}^{n+1}_{\alpha,\mathcal{R}} = \mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A}^n_{\alpha,\mathcal{R}})$ *for* $n \in \mathbb{N}$
- $\mathcal{A}^\star_{\alpha,\mathcal{R}} = \lim_{n \to +\infty} \mathcal{A}^n_{\alpha,\mathcal{R}}$

Note that even if $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ cannot be finitely computed in general, in many cases there exists a natural $k \in \mathbb{N}$ such that $\mathcal{A}^k_{\alpha,\mathcal{R}}$ is a fixpoint, i.e. $\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A}^k_{\alpha,\mathcal{R}}) = \mathcal{A}^k_{\alpha,\mathcal{R}}$. In the following proposition, we give some sufficient conditions for building an over-approximation automaton $\mathcal{B}$ of the set of $\mathcal{R}$-descendants of a regular language recognized by $\mathcal{A}$.

Definition 7 *Let $\mathcal{A}$ be an automaton and $\mathcal{R}$ a TRS, $\mathcal{R}$ and $\mathcal{A}$ satisfy the left-linearity condition if:*

$$\forall t \in \mathcal{T}(\mathcal{F}),\ \exists \tau : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F}),\ \exists l \to r \in \mathcal{R}\ \ s.t.\ \ t = l\tau \to^\star_\Delta q$$
$$\Rightarrow$$
$$\exists \sigma \in \Sigma(\mathcal{Q}, \mathcal{X})\ \ s.t.\ \ t \to^\star_\Delta l\sigma \to^\star_\Delta q$$

Note that for every left-linear TRS $\mathcal{R}$ trivially satisfy the left-linearity condition with any tree automaton. This condition is, in fact, necessary for non left-linear TRS. Roughly, the problem with non left-linear rules is the following: let $f(x,x) \to g(x)$ be a rule of $\mathcal{R}$ and let $\mathcal{A}$ be a tree automaton whose set of transitions contains $f(q_1,q_1) \to q_0$ and $f(q_2,q_3) \to q_0$. Although we can construct a valid substitution $\sigma = \{x \mapsto q_1\}$ for matching the rewrite rule on the first transition, it is not the case for the second one. The semantics of a completion between rule $f(x,x) \to g(x)$ and transition $f(q_2,q_3) \to q_0$ would be to find the common language of terms recognized both by q_2 and q_3. This can be obtained by computing a new tree automaton $\mathcal{A}'$ with a set of states $\mathcal{Q}'$ such that $\mathcal{Q}'$ is disjoint from states of $\mathcal{A}$ and $\exists q \in \mathcal{Q}' : \mathcal{L}(\mathcal{A}', q) = \mathcal{L}(\mathcal{A}, q_2) \cap \mathcal{L}(\mathcal{A}, q_3)$. Then, to end the completion step it would be enough to add transitions of $\mathcal{A}'$ to $\mathcal{A}$ with the new transition $g(q) \to q_0$.

On the other hand, one can remark that the non-linearity problem would disappear with deterministic automata since for any deterministic automaton $\mathcal{A}_{det}$ and for all states q, q' of

$\mathcal{A}_{det}$ we trivially have $\mathcal{L}(\mathcal{A},q) \cap \mathcal{L}(\mathcal{A},q') = \emptyset$. However, determinization of a tree automaton may result in an exponential blow-up of the number of states [5].

A solution, in between the two previous ones, is to use the *left-linearity condition* defined above by ensuring determinism for a subset of states $q \in \mathcal{Q}$ which are to be matched by the non linear variables of the non linear rules [2]. For instance, on the last example, it is enough to build the first critical pair, add the transition $g(q_1) \to q_0$, and keep q_2, q_3 deterministic, i.e. such that $\mathcal{L}(\mathcal{A}^\star_{\alpha,\mathcal{R}}, q_2) \cap \mathcal{L}(\mathcal{A}^\star_{\alpha,\mathcal{R}}, q_3) = \emptyset$. We now define this condition called *simple left-linearity* which implies the left-linearity condition. Let $\mathcal{A}$ be an automaton, $l \to r$ a rewrite rule over $\mathcal{T}(\mathcal{F},\mathcal{X})$, $\{x_1,\ldots,x_k\}$ the set of variables non linear in l and $\mathcal{Y}$ a set of variables distinct from $\mathcal{X}$. Let $Ren(l)$ be the pair (l',E) where l' denotes the term where non linear variables are renamed and E is a set of constraint.

$$
\begin{array}{lcll}
Ren(l) & = & (l,\emptyset) & \text{if } l \text{ is either a constant or a variable that does not appear in } \{x_1,\ldots,x_k\} \\
 & = & (y,\{x=y\}) & \text{if } l \text{ is a variable } x \in \{x_1,\ldots,x_k\} \text{ and } y \text{ if a fresh variable of } \mathcal{Y} \\
 & = & (f(t'_1,\ldots,t'_n), \bigcup_{i=1}^n E_i) & \text{if } l = f(t_1,\ldots,t_n) \text{ and } Ren(t_i) = (t'_i,E_i) \text{ for all } i = 1\ldots n.
\end{array}
$$

Definition 8 *(Simple left-linearity condition) An automaton $\mathcal{A}$ and a TRS $\mathcal{R}$ satisfy* simplified left-linearity condition *if for all rule $l \to r \in \mathcal{R}$ such that $Ren(l) = (l',E)$:*

$$\forall (x=y) \in E,\ \forall \sigma \in \Sigma(\mathcal{Q},\mathcal{X}),\ \forall q,q_x,q_y \in \mathcal{Q} :\ l'\sigma \to^\star_\Delta q \wedge \sigma(x) = q_x \neq q_y = \sigma(y)$$

$$\Longrightarrow$$

$$\mathcal{L}(\mathcal{A},q_x) \cap \mathcal{L}(\mathcal{A},q_y) = \emptyset$$

Proposition 1 *Simple left-linearity implies left-linearity.*

Proof 1 *If $\mathcal{A}$ does not verify the linearity condition induced by $l \to r$, there is at least a ground term t recognized by a state q of $\mathcal{A}$, a substitution $\tau : \mathcal{X} \to \mathcal{T}(\mathcal{F})$ such that $t = l\tau$ and $t \to^\star_\Delta q$, and there is no $\mathcal{Q}$-substitution $\sigma \in \Sigma(\mathcal{Q},\mathcal{X})$ such that $t \to^\star_\Delta l\sigma \wedge l\sigma \to^\star_\Delta q$. However, if t is an instance of l then t is also instance of l', let $t = l'\rho$ where $\rho : \mathcal{Y} \to \mathcal{T}(\mathcal{F})$. The problem is solved by case reasoning on t: t cannot be a variable a otherwise $l = a$ and $\mathcal{A}$ respects the linearity condition, then t is of the form $f(t_1,\ldots,t_n)$. Let $t = l'\rho'$ such that*

[2] This is what is called locally deterministic tree automata in [15].

$\rho' : \{y_1, \ldots, y_k\} \mapsto \mathcal{T}(\mathcal{F})$, we denotes $\rho'(y_j) = t_j$. If $t \to^\star_\Delta q$ than all subterm of t are recognized by $\mathcal{A}$ and there are k states $q_1, \ldots, q_k$ such that $t_j \to q_j$ for $1 \leq j \leq k$. We construct a $\mathcal{Q}$-substitution $\sigma' : \{y_1, \ldots, y_k\} \to \mathcal{Q}$ define by $\sigma'(y_j) = q_j$. We have $t \to^\star_\Delta l'\sigma'$ then $l'\sigma' \to^\star_\Delta q$, however by hypothesis, there is no $\mathcal{Q}$-substitution $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ such that $t \to^\star_\Delta l\sigma \wedge l\sigma \to^\star_\Delta q$, necessarily there are at least two variables y_i and y_j such that

1. *$\sigma'(y_i) = q_i$ and $\sigma'(y_j) = q_j$ with $q_i \neq q_j$*
2. *$y_i = y_j$ is a constraint of E*
3. *$t_i = t_j$*

The condition 1. and 2. hold true for at least a pair of variables (y_i, y_j) otherwise we could construct a $\mathcal{Q}$-substitution σ such that $l\sigma = l'\sigma'$. The condition 3. holds true because t is an instance of l. This leads to a contradiction.

Proposition 2 *Let $\mathcal{R}$ be a TRS, $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, and $B = \langle \mathcal{F}, \mathcal{Q}', \mathcal{Q}_f, \Delta' \rangle$ two tree automata such that $\mathcal{R}$ and $\mathcal{B}$ satisfy the left-linearity condition. We have $\mathcal{R}^\star(\mathcal{L}(A)) \subseteq \mathcal{L}(B)$ if*

1. *$\Delta \subseteq \Delta'$, and*
2. *$\forall l \to r \in \mathcal{R}$, $\forall q \in \mathcal{Q}'$, $\forall \sigma \in \Sigma(\mathcal{Q}', \mathcal{X})$, $l\sigma \to^\star_{\Delta'} q$ implies $r\sigma \to^\star_{\Delta'} q$.*

Proof 2 *By definition, any term t of $\mathcal{R}^\star(\mathcal{L}(A))$ is such that $\exists s \in \mathcal{L}(A)$ s.t. $s \to^\star_\mathcal{R} t$. By induction on the size of the derivation $s \to^\star_\mathcal{R} t$, we prove that if $s \to^\star_\mathcal{R} t$ and $s \to^\star_{\Delta'} q$ with $q \in \mathcal{Q}_f$ then $t \to^\star_{\Delta'} q$, which implies that $t \in \mathcal{L}(B)$.*

1. *if $t = s$ then, since $s \in \mathcal{L}(A)$, we have that $\exists q \in \mathcal{Q}_f$ s.t. $t = s \to^\star_\Delta q$. Moreover, $\Delta \subseteq \Delta'$, hence $\exists q \in \mathcal{Q}_f$ s.t. $t \to^\star_{\Delta'} q$,*
2. *if $s \to^+_\mathcal{R} t$, then $\exists s' \in \mathcal{T}(\mathcal{F})$ s.t. $s \to^\star_\mathcal{R} s' \to_\mathcal{R} t$. By induction hypothesis applied to $s \to^\star_\mathcal{R} s'$, we obtain that $\exists q \in \mathcal{Q}_f$ s.t. $s' \to^\star_{\Delta'} q$. Moreover, since $s' \to_\mathcal{R} t$, there exists a rule $l \to r \in \mathcal{R}$, a substitution τ, and a position p in s' such that $l\tau = s'|_p$ and $t = s'[r\tau]_p$. By construction of bottom-up tree automata with normalized transitions, if $s' \to^\star_{\Delta'} q$, then any subterm of s' is reducible by Δ' into a state of $\mathcal{Q}'$. Hence, since $l\tau = s'|_p$, we get that $\exists q' \in \mathcal{Q}'$ s.t. $l\tau \to^\star_{\Delta'} q'$ and $s'[q']_p \to^\star_{\Delta'} q$. Now, let us show that $r\tau \to^\star_{\Delta'} q'$. Let $\mathcal{V}ar(l) = \{x_1, \ldots, x_k\}$ be the variables of l. Since $\mathcal{R}$ and $\mathcal{B}$ satisfy the left-linearity condition, we get that there exists $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ such that $l\tau \to^\star_{\Delta'} l\sigma \to^\star_{\Delta'} q'$. Hence, there exists some states $q \in \mathcal{Q}$ such that $\sigma = \{x_i \mapsto q_i \mid i = 1 \ldots k\}$ and $x_i\tau \to^\star_{\Delta'} q_i$ for $i = 1 \ldots k$. From $x_i\tau \to^\star_{\Delta'} q_i$ we get that $r\tau \to^\star_{\Delta'} r\sigma$. Finally, since $r\sigma \to^\star_{\Delta'} q'$, we get that $r\tau \to^\star_{\Delta'} q'$ and thus $t = s'[r\tau]_p \to^\star_{\Delta'} q$*

In this first theorem, we show that completion always over-approximate the set of descendants for TRSs and tree automata satisfying the left-linearity condition.

Theorem 1 *Given a tree automaton $\mathcal{A}$ and a TRS $\mathcal{R}$ satisfying the left-linearity condition, for any abstraction function α,*

$$\mathcal{L}(\mathcal{A}^\star_{\alpha,\mathcal{R}}) \supseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$$

Proof 3 *For proving $\mathcal{L}(\mathcal{A}^\star_{\alpha,\mathcal{R}}) \supseteq \mathcal{R}^\star\mathcal{L}(A)$, it is enough to prove that the approximation automata verifies Conditions 1 and 2 of Proposition 2, for all abstraction function α. By Definition 6, $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ trivially verifies Condition 1. Now, to prove that $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ also verifies Condition 2 of Proposition 2, it is enough to prove that $Norm_\alpha(r\sigma \to q) \subseteq \Delta'$ implies $r\sigma \to^\star_{\Delta'} q$.*

Let s' be any subterm of $r\sigma$ (possibly non-strict) and $q' \in \mathcal{Q}'$. By induction on the size of s', we show that $Norm_\alpha(s' \to q') \subseteq \Delta'$ implies that $s' \to^\star_{\Delta'} q'$:

- *if $s' = q'$, then we trivially have $s' \to^\star_{\Delta'} q'$.*
- *if $s' = q'' \in \mathcal{Q}'$ s.t. $q'' \neq q'$ then, by case 2 of definition of $Norm$, we get that $Norm_\alpha(s' \to q') = \{s' \to q'\}$. Since $Norm_\alpha(s' \to q') \subseteq \Delta'$, we have $s' \to^\star_{\Delta'} q'$.*
- *if $s' = g(t_1, \ldots, t_m) \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}')$, by applying case 3 of definition of $Norm$, we get that*

  (a) $\{g(top_\alpha(t_1), \ldots, top_\alpha(t_m)) \to q'\} \subseteq \Delta'$, and

  (b) $\bigcup_{i=1}^n Norm_\alpha(t_i \to top_\alpha(t_i)) \subseteq \Delta'$,

  where $\forall i = 1 \ldots n$, $top_\alpha(t_i) \in \mathcal{Q}'$. By applying induction hypothesis to (b), we get that $\forall i = 1 \ldots n$, $t_i \to^\star_{\Delta'} top_\alpha(t_i)$. On the other hand, (a) implies that $g(top_\alpha(t_1), \ldots, top_\alpha(t_m)) \to_{\Delta'} q'$. As a result, $g(t_1, \ldots, t_m) \to^\star_{\Delta'} g(top_\alpha(t_1), \ldots, top_\alpha(t_m)) \to_{\Delta'} q'$.

Hence $Norm_\alpha(r\sigma \to q) \subseteq \Delta'$ implies $r\sigma \to^\star_{\Delta'} q$, and Condition 2 of Proposition 2 is satisfied by $\mathcal{A}^\star_{\alpha,\mathcal{R}}$.

4 The exact case

The aim of this part is to refine the previous result and show that known regular classes of descendants can be computed using the tree automata completion algorithm and some particular abstraction functions. In a first part, we give some sufficient conditions on the abstraction function α so that completion is exact w.r.t. $\mathcal{R}^\star(E)$. Then we will see how all regular classes of the literature can be expressed using abstraction functions satisfying those conditions.

Definition 9 *(Right-linearity condition) A TRS $\mathcal{R}$ and tree automaton $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ satisfy the right-linearity condition if*

1. *$\mathcal{R}$ is right-linear, or*

2. $\forall q \in \mathcal{A} : \exists t \in \mathcal{T}(\mathcal{F}) : \mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{R}^\star(t)$

Let us show that those two conditions are necessary for exactness of completed automata on a counterexample.

Example 2 *Let $\mathcal{R} = \{f(x) \to g(x,x)\}$ be a non right-linear TRS and let $\mathcal{A}$ be the tree automaton such that $\mathcal{Q}_f = \{q_0\}$ and $\Delta = \{f(q_1) \to q_0, a \to q_1, b \to q_1\}$. Note that $\mathcal{A}$ is deterministic and that $\mathcal{L}(\mathcal{A}) = \{f(a), f(b)\}$ is finite. However, the completed automaton $\mathcal{A}^1_{\alpha,\mathcal{R}}$ (for any abstraction function α) has a new transition $g(q_1, q_1) \to q_0$ and the recognized language becomes $\mathcal{L}(\mathcal{A}^n_{\alpha,\mathcal{R}}) = \{f(a), g(a,a), g(a,b), g(b,a), g(b,b)\}$ which is a* superset *of $\mathcal{R}^\star(\mathcal{L}(\mathcal{A})) = \{f(a), g(a,a), g(b,b)\}$. However, if $\mathcal{R}$ was linear or if there was no transition $b \to q_1$ in Δ then $\mathcal{A}^n_{\alpha,\mathcal{R}}n$ would have been different and would have been computed exactly. Similarly, if we add a rule $a \to b$ to $\mathcal{R}$ then $\mathcal{A}^1_{\alpha,\mathcal{R}}$ recognize exactly $\mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$ and right-linearity condition is satisfied since for $q_1 \in \mathcal{A}$ there exists the term a such that $\mathcal{L}(\mathcal{A}, q_1) \subseteq \mathcal{R}^\star(\{a\}) = \{a, b\}$.*

Note that this condition focuses only on the *initial* automaton and not on the completed one. Hence, this condition is trivially satisfied (using the second case) by any deterministic tree automaton recognizing a finite language. This will be useful in the following theorems for defining regular classes of $\mathcal{R}^\star(E)$ for finite sets E.

Lemma 1 *Let $\mathcal{R}$ be a TRS, $l \to r \in \mathcal{R}$ be a rewrite rule with $r \notin \mathcal{X}$ and $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ a tree automaton without dead states. Let $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ be a $\mathcal{Q}$-substitution, such that $l\sigma \to r\sigma$ and $l\sigma \to^\star_{\mathcal{A}} q$ with $q \in \mathcal{Q}$.*

For all $t \in \mathcal{T}(\mathcal{F})$ s.t. $t \to^\star_{\mathcal{A}} r\sigma$ there exists $\delta : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$ s.t. $l\delta \in \mathcal{T}(\mathcal{F})$, $l\delta \to_{\mathcal{A}}{}^\star q$ and $l\delta \to^\star_{\mathcal{R}} r\delta = t$, if $\mathcal{R}$ and $\mathcal{A}$ satisfy the right-linearity condition.

Proof 4 *Let $\{p_1, \ldots, p_n\} = \mathcal{P}os_{\mathcal{X}}(r)$ and $\forall i = 1 \ldots n : x_i = r|_{p_i}$. Note that if there exists p_i and p_j s.t. $r|_{p_i} = r|_{p_j}$ then $x_i = x_j$. Let $\{y_1, \ldots, y_m\} = \mathcal{V}ar(l) \setminus \mathcal{V}ar(r)$ be the set of variables of l that do not occur in r. Note that for l it is not necessary to distinguish the multiple occurrences of non linear variables. Let $q_1, \ldots, q_n, q'_1, \ldots, q'_m \in \mathcal{Q}$ be the states such that $\sigma = \{x_1 \mapsto q_1, \ldots x_n \mapsto q_n, x'_1 \mapsto q'_1, \ldots, x'_m \mapsto q'_m\}$ and $q_i = q_j$ for all $i, j \in \{1, \ldots, n\}$ such that $x_i = x_j$. Thus $r = r[x_1, \ldots, x_n]$ and $r\sigma = r[q_1, \ldots, q_n]$. On the other hand, by construction of tree automata $t \to^\star_{\mathcal{A}} r\sigma = r[q_1, \ldots, q_n]$ implies that there exists $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ such that $t = r[t_1, \ldots, t_n]$ and $t_1 \to^\star_{\mathcal{A}} q_1, \ldots, t_n \to^\star_{\mathcal{A}} q_n$.*

Let δ_r be the relation $\{x_1 \mapsto, \ldots, x_n \mapsto t_n\}$. Note that δ_r is a substitution (i.e. a function) if and only if there is no $i, j \in \{1, \ldots, n\}$ such that $x_i = x_j$ and $t_i \neq t_j$. This is of course trivially the case if r is linear. Otherwise, we may have $t_i \to^\star_{\mathcal{A}} q_i$, $t_j \to^\star_{\mathcal{A}} q_j$, $q_i = q_j$ but $t_i \neq t_j$ and thus δ_r would not be a function. However, if condition 2. of definition 9 is satisfied then every term which is recognized into q_i is either t'_i or one of its descendants (i.e. $t_i, t_j \in \mathcal{R}^\star(t'_i)$). In this case, if we replace t_i and t_j by t'_i in δ_r (and proceed similarly for every other occurrence of a non linear variable), we obtain a valid substitution δ_r such that $r\delta_r \to_{\mathcal{R}}{}^\star r[t_1, \ldots, t_n]$. Thus, using case 1. or case 2. of definition 9 leads

to the same property: we have built a substitution δ_r such that $r\delta_r \rightarrow_{\mathcal{R}}^{\star} r[t_1, \ldots, t_n]$ and $r[t_1, \ldots, t_n] \rightarrow_{\mathcal{A}}^{\star} r\sigma$.

Now, let $\delta = \delta_r \cup \delta_l$ where $\delta_l = \bigcup_{i=1\ldots m}\{y_i \mapsto u_i' \mid \exists u_i' \in \mathcal{T}(\mathcal{F})$ s.t. $u_i' \rightarrow_{\mathcal{A}}^{\star} q_i'$ and $q_i' = \sigma(y_i)\}$. Note that the existence of u_i' s.t. $u_i' \rightarrow_{\mathcal{A}}^{\star} q_i'$ is guaranteed by the fact that q_i' is a state of $\mathcal{A}$ and there is no dead state in $\mathcal{A}$. The relation δ_l is a substitution and so is δ_r. Furthermore, since $\mathcal{D}om(\delta_r) \cap \mathcal{D}om(\delta_l) = \emptyset$ then δ is a substitution. Finally. we have $l\delta \in \mathcal{T}(\mathcal{F})$, $l\delta \rightarrow_{\mathcal{A}}^{\star} l\sigma$ and $\delta \rightarrow_R^{\star} r\delta \rightarrow_R^{\star} r[t_1, \ldots, t_n] = t$.

We now introduce *coherent abstraction function* which define some subclasses of completion algorithms for which the automaton completion algorithm is exact. Informally, an abstraction function is coherent with regards to a tree automaton $\mathcal{A}$ and a term rewriting system $\mathcal{R}$ if for every configuration t and every state q such that α maps t to q, either q is not a state of $\mathcal{A}$ (it is a new state) or terms recognized by q in $\mathcal{A}$ are either a term t' recognized by t (i.e. $t' \rightarrow_{\mathcal{A}}^{\star} t$) or $\mathcal{R}$-descendants of t'.

Definition 10 *(Coherent abstraction function) Let $\mathcal{R}$ be a TRS, $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton and α be an abstraction function. The function α is said to be* coherent *with $\mathcal{R}$ and $\mathcal{A}$ if for all $t \in \mathcal{D}om(\alpha)$, for all $q \in \mathcal{Q} \cap \mathcal{R}an(\alpha)$ if $\alpha(t) = q$ then $t \rightarrow q \in \mathcal{A}$ and $\exists t'$ s.t. $\mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{R}^{\star}(\{t' \mid t' \rightarrow_{\mathcal{A}}^{\star} t\})$.*

Lemma 2 *Let $\mathcal{R}$ be a TRS, $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton and α be an abstraction function. If $\mathcal{R}$ and $\mathcal{A}$ satisfy the right-linearity condition and if α is injective and coherent with regards to $\mathcal{R}$ and $\mathcal{A}$ then:*

$$\forall t \in \mathcal{T}(\mathcal{F}), \forall q \in \mathcal{Q} : t \in \mathcal{L}(\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A}), q) \implies t \in \mathcal{R}^{\star}(\mathcal{L}(A, q))$$

Proof 5 *For terms t such that $t \in \mathcal{L}(\mathcal{A}, q)$, we trivially have that $t \in \mathcal{R}^{\star}(\mathcal{L}(\mathcal{A}, q))$. So, we can restrict the proof to terms t such that $t \notin \mathcal{L}(\mathcal{A}, q)$. Similarly, we can distinguish another particular case where $t \in \mathcal{L}(\mathcal{A}, q')$, $t \notin \mathcal{L}(\mathcal{A}, q)$ and $q' \rightarrow q \in \mathcal{C}_{\alpha,\mathcal{R}}$. In that case, the completion step producing $\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})$ from $\mathcal{A}$ necessarily builds a critical pair of the form $l\sigma \rightarrow_{\mathcal{R}} r\sigma = q'$ and $l\sigma \rightarrow_{\mathcal{A}}^{\star} q$ where $l \rightarrow r \in \mathcal{R}$. In that case, we necessarily have $l = C[x]$ and $r = x$ where $x \in \mathcal{V}ar()$ and $\sigma = \{x \mapsto q\} \cup \sigma'$. Hence, we have $l\sigma = C[q']\sigma' \rightarrow_{\mathcal{A}}^{\star} q$ and since $t \in \mathcal{L}(\mathcal{A}, q')$, we have $C[t] \rightarrow_{\mathcal{A}}^{\star} C[q'] \rightarrow_{\mathcal{A}}^{\star} q$ and thus $C[t] \in \mathcal{L}(\mathcal{A}, q)$. Finally since the rule $l \rightarrow r$ is of the form $C[x] \rightarrow x$ we get that $t \in \mathcal{R}^{\star}(\mathcal{L}(\mathcal{A}, q))$.*

Now for other cases, we proceed by induction over the height of t.

- *If height of t is 0 then $t = a$ where a is a constant. Since $a \in \mathcal{L}(\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A}), q)$ we have $a \rightarrow q \in \mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})$. Since $a \notin \mathcal{A}$ then the completion step producing $\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})$ from $\mathcal{A}$ necessarily builds a critical pair of the form $l\sigma \rightarrow_{\mathcal{R}} a$ and $l\sigma \rightarrow_{\mathcal{A}}^{\star} q$ where $l \rightarrow a \in \mathcal{R}$. By lemma 1, we obtain that there exists a substitution δ such that $l\delta \in \mathcal{T}(\mathcal{F})$, $l\delta \rightarrow_{\mathcal{A}}^{\star} q$ and $l\delta \rightarrow_{\mathcal{R}} a$, hence $a \in \mathcal{R}^{\star}(\mathcal{L}(\mathcal{A}, q))$.*

- *Now, we assume that the property is true for terms of height n. Let us prove that the property also holds for terms of height $n+1$. Let t be a term of height $n+1$ such that $t \in \mathcal{L}(\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A}), q)$. Let $f \in \mathcal{F}$ and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F})$ terms of height lesser or equal to n such that $t = f(t_1, \ldots, t_n)$. By construction of tree automata, $t \in \mathcal{L}(\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A}), q)$ implies that there exists states $q_1, \ldots, q_n$ such that $f(t_1, \ldots, t_n) \rightarrow^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} f(q_1, \ldots, q_n) \rightarrow^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} q$. Then by case on $f(q_1, \ldots, q_n) \rightarrow q \in \mathcal{A}$, we obtain:*

  - *Assume that $f(q_1, \ldots, q_n) \rightarrow q \in \mathcal{A}$. Then $q_1, \ldots, q_n \in \mathcal{A}$ and by induction hypothesis we get that $t_i \in \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q_i))$ for $i = 1 \ldots n$. Hence, there exists terms t'_i such that $t'_i \rightarrow_\mathcal{A}^\star q_i$ and $t'_i \rightarrow_\mathcal{R} t_i$. Hence $f(t'_1, \ldots f'_n) \rightarrow_\mathcal{A}^\star f(q_1, \ldots, q_n) \rightarrow_\mathcal{A} q$ and $f(t'_1, \ldots f'_n) \rightarrow_\mathcal{R}^\star f(t_1, \ldots, t_n)$, i.e. $f(t'_1, \ldots, f'_n) \in \mathcal{L}(\mathcal{A}, q)$ and $t \in \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q))$.*
  - *Now, assume that $f(q_1, \ldots, q_n) \rightarrow q \notin \mathcal{A}$. Thus, transition $f(q_1, \ldots, q_n) \rightarrow q$ has been added to $\mathcal{A}$ by the completion step: there is either a critical pair of the form* **(a)** *$l\sigma \rightarrow_\mathcal{R} C[f(t''_1, \ldots, t''_n)]$ and $l\sigma \rightarrow_\mathcal{A}^\star q'$ or* **(b)** *$l\sigma \rightarrow_\mathcal{R} f(t''_1, \ldots, t''_n)$ and $l\sigma \rightarrow_\mathcal{A}^\star q$. Let us continue the proof on those two cases:*

    (a) *Assume that there is a critical pair of the form $l\sigma \rightarrow_\mathcal{R} C[f(t''_1, \ldots, t''_n)]$ and $l\sigma \rightarrow_\mathcal{A}^\star q'$. In order to produce the new transition $f(q_1, \ldots, q_n) \rightarrow q$ i.e. to have $f(q_1, \ldots, q_n) \rightarrow q \in Norm_\alpha(C[f(t''_1, \ldots, t''_n)] \rightarrow q')$, it is necessary to have $top_\alpha(f(t''_1, \ldots, t''_n)) = q$ and $\forall i = 1 \ldots n : top_\alpha(t''_i) = q_i$. However, since $q \in \mathcal{A}$ and α is coherent with $\mathcal{R}$ and $\mathcal{A}$, we get that $Norm_\alpha(f(t''_1, \ldots, t''_n) \rightarrow q) \subseteq \mathcal{A}$, hence $f(q_1, \ldots, q_n) \rightarrow q \in \mathcal{A}$ which contradicts $f(q_1, \ldots, q_n) \rightarrow q \notin \mathcal{A}$*

    (b) *Assume that there is a critical pair of the form $l\sigma \rightarrow_\mathcal{R} r\sigma = f(t''_1, \ldots, t''_n)$ and $l\sigma \rightarrow_\mathcal{A}^\star q$. First, let us prove that $\forall i = 1 \ldots n : t_i \rightarrow^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} t''_i$. We already now that $t_i \rightarrow^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} q_i$. By cases on $q_i \in \mathcal{A}$, we obtain:*

    - *if q_i occurs in Δ (there is at least one transition $c \rightarrow q_i$ in Δ) then since α is coherent with $\mathcal{R}$ and $\mathcal{A}$ and $top_\alpha(t''_i) = q_i$, we get that $\mathcal{L}(\mathcal{A}, q_i) \subseteq \mathcal{R}^\star(\{s \mid s \rightarrow_\mathcal{A}^\star t''_i\})$. Hence, since $t_i \rightarrow_\mathcal{A}^\star q_i$ we get that there exists a term s_i such that $s_i \rightarrow_\mathcal{A}^\star t''_i$ and $s_i \rightarrow_\mathcal{R}^\star t_i$.*
    - *if q_i does not occur in Δ then the only rewriting path from t_i to q_i is necessarily $t_i \rightarrow^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} t''_i \rightarrow^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} q_i$, hence $t_i \rightarrow^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} t''_i$. In that case let $s_i = t_i$.*

    Thus, we have $f(s_1, \ldots, s_n) \rightarrow_\mathcal{A}^\star f(t''_1, \ldots, t''_n) \rightarrow_\mathcal{A}^\star q$ and $f(s_1, \ldots, s_n) \rightarrow_\mathcal{R}^\star f(t_1, \ldots, t_n)$. Finally, applying lemma 1 on term $f(s_1, \ldots, s_n)$, we obtain that for $f(s_1 \ldots, s_n)$ such that $f(s_1, \ldots, s_n) \rightarrow^\star_{\mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{A})} r\sigma$ there exists δ such that $r\delta = f(s_1, \ldots, s_n)$, $l\delta \in \mathcal{T}(\mathcal{F})$, $l\delta \rightarrow_\mathcal{A}^\star l\sigma \rightarrow_\mathcal{A}^\star q$ and $l\delta \rightarrow_\mathcal{R}^\star r\delta$. Hence, we have a term $l\delta$ such that $l\delta \rightarrow_\mathcal{A}^\star q$ and $l\delta \rightarrow_\mathcal{R}^\star r\delta \rightarrow_\mathcal{R}^\star f(t_1, \ldots, t_n)$.

Theorem 2 *Let $\mathcal{R}$ be a TRS, $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ a tree automaton and α an injective abstraction function coherent with $\mathcal{R}$ and $\mathcal{A}$. If $\mathcal{R}$ and $\mathcal{A}$ satisfy the right-linearity condition then*

$$\forall n \in \mathbb{N} : \mathcal{L}(\mathcal{A}^n_{\alpha,\mathcal{R}}) \subseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$$

Proof 6 *We proceed by induction on n. If $n = 0$ we have $\mathcal{A}^0_{\alpha,\mathcal{R}} = \mathcal{A}$ and thus $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$. Then, we assume that the property holds for n and we prove that it holds for $n+1$. Let us denote by $\mathcal{B}$ the tree automaton $\mathcal{A}^1_{\alpha,\mathcal{R}}$. Then, the proof is done by using the induction hypothesis on $\mathcal{B}$ since we have $\mathcal{A}^{n+1}_{\alpha,\mathcal{R}} = \mathcal{C}_{\alpha,\mathcal{R}}(\mathcal{B}^n_{\alpha,\mathcal{R}})$. By lemma 2, we know that for every state $q \in \mathcal{Q}$, and for every term $t \in \mathcal{L}(\mathcal{B}, q)$ we have $t \in \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q))$. This property is true in particular for final states, thus we have: $\mathcal{L}(\mathcal{B}) \subseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$. Now, in order to use the induction hypothesis, we need to prove that* **(a)** *α is coherent with $\mathcal{R}$ and $\mathcal{B}$ and that* **(b)** *$\mathcal{R}$ and $\mathcal{B}$ satisfy the right-linearity condition.*

(a) *For every normalized configuration $t \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ such that $\alpha(t) = q$,*

- *if $q \in \mathcal{A}$ then we had already $t \to q \in \mathcal{A}$ so $t \to q \in \mathcal{B}$. Since α is coherent with $\mathcal{R}$ and $\mathcal{A}$, we know that $\mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{R}^\star(\{t' \mid t' \to_\mathcal{A}{}^\star\ t\})$. By applying the $\mathcal{R}^\star$ operator to both sides of the previous inequality, we obtain that $\mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q)) \subseteq \mathcal{R}^\star(\mathcal{R}^\star(\{t' \mid t' \to_\mathcal{A}{}^\star\ t\}))$. On the other hand, by lemma 2, we get that $\mathcal{L}(\mathcal{B}, q) \subseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{A}, q))$. Note that $\mathcal{R}^\star(\mathcal{R}^\star(E)) = \mathcal{R}^\star(E)$ for any set E, hence we have: $\mathcal{L}(\mathcal{B}, q) \subseteq \mathcal{R}^\star(\{t' \mid t' \to_\mathcal{A}{}^\star\ t\})$. Moreover, since $\{t' \mid t' \to_\mathcal{A}{}^\star\ t\} \subseteq \{t' \mid t' \to^\star_\mathcal{B} t\}$ we have $\mathcal{R}^\star(\{t' \mid t' \to_\mathcal{A}{}^\star\ t\}) \subseteq \mathcal{R}^\star(\{t' \mid t' \to^\star_\mathcal{B} t\})$ and by transitivity of $\subseteq$, we get that $\mathcal{L}(\mathcal{B}, q) \subseteq \mathcal{R}^\star(\{t' \mid t' \to^\star_\mathcal{B} t\})$.*
- *if $q \notin \mathcal{A}$ but $q \in \mathcal{B}$ then q is a state that has been introduced in the last completion step and since α is injective we know that t is the unique normalized configuration s.t. $t \to_\mathcal{B}{}^\star\ q$. Hence, $\mathcal{L}(\mathcal{B}, q) = \{t' \mid t' \to_\mathcal{B}{}^\star\ t\} \subseteq \mathcal{R}^\star(\{t' \mid t' \to_\mathcal{B}{}^\star\ t\})$.*

(b) *We know initially that $\mathcal{R}$ and $\mathcal{A}$ satisfy the right-linearity condition. If $\mathcal{R}$ and $\mathcal{A}$ satisfy the condition because $\mathcal{R}$ is right-linear then it will clearly be the case for $\mathcal{R}$ and $\mathcal{B}$. Otherwise, we know that $\forall q \in \mathcal{A} : \exists t \in \mathcal{T}(\mathcal{F}) : \mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{R}^\star(t)$ and we have to prove that it is also the case for $\mathcal{B}$.*

- *if $q \in \mathcal{A}$ then we know that $\exists t \in \mathcal{T}(\mathcal{F}) : \mathcal{L}(\mathcal{A}, q) \subseteq \mathcal{R}^\star(t)$. From lemma 2 we get that every term recognized by q in $\mathcal{B}$ has an ancestor in the terms recognized by q in $\mathcal{A}$, i.e. $\forall t_\mathcal{B} \in \mathcal{L}(\mathcal{B}, q) : \exists t_\mathcal{A} \in \mathcal{L}(\mathcal{A}, q)$ s.t. $t_\mathcal{A} \to_\mathcal{R}{}^\star\ t_\mathcal{B}$. Since, every term recognized by q in $\mathcal{A}$ has a common ancestor t, it is also the case for terms recognized by q in $\mathcal{B}$ (and it is the same ancestor t), i.e. $t \to_\mathcal{R}{}^\star\ t_\mathcal{A} \to_\mathcal{R}{}^\star\ t_\mathcal{B}$. Hence, $\forall q \in \mathcal{B} : \exists t \in \mathcal{T}(\mathcal{F}) : \mathcal{L}(\mathcal{B}, q) \subseteq \mathcal{R}^\star(t)$.*

- *if $q \notin \mathcal{A}$ but $q \in \mathcal{B}$ then q is a state that has been introduced in the last completion step. Let $s \to q'$ be the new transition whose normalization has led to construction of state q, i.e. $s = C[u]$ and $top_\alpha(u) = q$. By induction on the height of u we show that $\exists t \in \mathcal{T}(\mathcal{F}) : \mathcal{L}(\mathcal{B}, q) \subseteq \mathcal{R}^\star(t)$:*
  - *if u is a constant, since $q \notin \mathcal{A}$ and α is injective we know that $u \to q$ is the unique transition with q on the right-hand side, hence $\mathcal{L}(\mathcal{B}, q) = \{u\} \subseteq \mathcal{R}^\star(u)$.*
  - *if $u = f(t_1, \ldots, t_n)$ then since α is injective we know that there is a unique normalized configuration $f(q_1, \ldots, q_n)$ such that $f(q_1, \ldots, q_n) \to_\mathcal{B} q$ and $t_i \to_\mathcal{B}^\star q_i$ for $i = 1 \ldots n$. For every state q_i, $i = 1 \ldots n$, it is possible to find a unique term t'_i such that $\mathcal{L}(\mathcal{B}, q_i) \subseteq \mathcal{R}^\star(t'_i)$. If $q_i \notin \mathcal{A}$ then we use the induction hypothesis on transition $t_i \to q_i$. Otherwise, if $q_i \in \mathcal{A}$, then the proof is similar to the first case of the proof: by hypothesis we know that $\mathcal{L}(\mathcal{A}, q_i) \subseteq \mathcal{R}^\star(t'_i)$ and from Lemma 2, we can lift this property to $\mathcal{B}$, i.e. $\mathcal{L}(\mathcal{B}, q_i) \subseteq \mathcal{R}^\star(t'_i)$. Finally $\mathcal{L}(\mathcal{B}, q) \subseteq \mathcal{R}^\star(f(t'_1, \ldots, t'_n))$*

Finally, applying the induction hypothesis to $\mathcal{B}$, we get that $\mathcal{A}^{n+1}_{\alpha,\mathcal{R}} = \mathcal{B}^n_{\alpha,\mathcal{R}} \subseteq \mathcal{R}^\star(\mathcal{L}(\mathcal{B})) \subseteq \mathcal{R}^\star(\mathcal{R}^\star(\mathcal{L}(\mathcal{A}))) = \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$.

Theorem 3 *Let $\mathcal{R}$ be a TRS, $\mathcal{A}$ be a tree automaton, α be an injective abstraction function coherent with $\mathcal{R}$ and $\mathcal{A}$.*

$$\mathcal{L}(\mathcal{A}^\star_{\alpha,\mathcal{R}}) = \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$$

if $\mathcal{R}$ and $\mathcal{A}$ fulfill the right-linearity condition and if $\mathcal{R}$ and $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ fulfill the left-linearity condition.

Proof 7 *Direct consequence of theorems 2 and 1.*

This theorem states the general properties of $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ but it says nothing about the existence of a finite $\mathcal{A}^\star_{\alpha,\mathcal{R}}$, i.e. of termination of the completion. In the following, we give some interesting instances of this theorem as corollaries and some conditions for completion to terminate. The two first corollaries permits to use automata completion as a rewriting tool: for any given finite initial language, tree automata completion produces every possible reachable term. We will show in section 7.3 that using tree automata completion in this setting provide an efficient alternative to breadth-first search for a particular descendant.

Corollary 1 *Let $\mathcal{R}$ be a TRS, $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton such that $\forall q \in \mathcal{Q} : Card(\mathcal{L}(\mathcal{A}, q)) = 1$, α an injective abstraction such that $\mathcal{R}an(\alpha) \cap \mathcal{Q} = \emptyset$.*

$$\mathcal{L}(\mathcal{A}^\star_{\alpha,\mathcal{R}}) = \mathcal{R}^\star(\mathcal{L}(\mathcal{A}))$$

if $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ and $\mathcal{A}$ satisfy the left-linearity condition.

Proof 8 *Consequence of Theorem 3. Since, $\mathcal{A}$ satisfy $\forall q \in \mathcal{Q} : Card(\mathcal{L}(\mathcal{A}, q)) = 1$, the right-linearity condition is trivially fulfilled. Similarly, since $\mathcal{R}an(\alpha) \cap \mathcal{Q} = \emptyset$, α is trivially coherent with $\mathcal{R}$ and $\mathcal{A}$.*

A direct consequence of this corollary is that applying completion to a tree automaton recognizing one term models exactly rewriting if α is injective and $\mathcal{R}$ is left-linear. Now, let us show that if any of the above restriction is not satisfied then the completed automaton no longer recognizes exactly the set of reachable terms.

Example 3 *(Left-linearity condition is necessary) Let $\mathcal{R} = \{f(x,x) \to g(x),\ a \to b\}$, let $\mathcal{A}$ be the tree automaton with $\mathcal{Q} = \{q_0, q_1, q_2\}$ and set of transitions $\Delta = \{f(q_1,q_2) \to q_0,\ a \to q_1,\ b \to q_2\}$. Note that $\mathcal{A}$ is deterministic, it recognize a finite language $\mathcal{L}(\mathcal{A}) = \{f(a,b)\}$ and it satisfies $\forall q \in \mathcal{Q} : Card(\mathcal{L}(\mathcal{A}, q)) = 1$. However, tree automata completion produces a unique new transition: $b \to q_1$ and the completed automaton does not recognize term $g(b)$ which is a descendant of $f(a,b)$.*

Note that, using determinisation after each step of completion would solve this problem and would led to a stronger corollary with no restriction on $\mathcal{R}$ and no left-linearity condition checking. However, as it was pointed out above, we also focus here on efficiency of algorithms and this is solution is not usable in practice since it adds an exponential overhead. Verifying the left-linearity condition (in particular simple left-linearity condition) is easier and is sufficient in many practical cases (see section 6.3).

Example 4 *(Unicity of initial recognized language is necessary) Let $\mathcal{R} = \{f(x) \to g(x,x)\}$, let $\mathcal{A}$ be the tree automaton with $\mathcal{Q} = \{q_0, q_1\}$ and set of transitions $\Delta = \{f(q_1) \to q_0,\ a \to q_1,\ b \to q_1\}$. Note that $\mathcal{A}$ is deterministic and recognize a finite language $\mathcal{L}(\mathcal{A}) = \{f(a), f(b)\}$. However, completion produces a new transition $g(q_1,q_1) \to q_0$ and thus the completed automaton recognizes terms $g(a,b)$ and $g(b,a)$ which are not valid descendants of $f(a)$ nor of $f(b)$.*

Example 5 *(Abstraction function needs to be injective) Let $\mathcal{R} = \{f(a) \to g(a,b)\}$, let $\mathcal{A}$ be the tree automaton with $\mathcal{Q} = \{q_0, q_1\}$ and set of transitions $\Delta = \{f(q_1) \to q_0,\ a \to q_1\}$. Note that $\mathcal{A}$ is deterministic, it recognizes a finite language $\mathcal{L}(\mathcal{A}) = \{f(a)\}$ and it satisfies $\forall q \in \mathcal{Q} : Card(\mathcal{L}(\mathcal{A}, q)) = 1$. However, tree automata completion produces a new transition $g(a,b) \to q_0$ which has to be normalized. Now assume that $\alpha = \{a \mapsto q_2,\ b \mapsto q_2\}$, then $Norm_\alpha(g(a,b) \to q_0) = \{a \to q_2,\ b \to q_2,\ g(q_2,q_2) \to q_0\}$. Thus, the completed automaton recognizes terms $g(a,a)$, $g(b,b)$ and $g(b,a)$ which are not valid descendants of $f(a)$.*

As we will see in section 5 with non regular sets of descendants, using non injective abstraction functions is a very convenient way to force completion to terminate and build over-approximations.

The following corollary will be used to give alternative proofs of results for ground TRSs [7, 3], linear and semi-monadic [6], linear and "decreasing" TRSs [18].

Corollary 2 *Let $\mathcal{R}$ be a linear TRS, $\mathcal{A}$ be a tree automaton, α an injective abstraction such that $\mathcal{R}an(\alpha) \cap \mathcal{Q} = \emptyset$*

$$\mathcal{L}(\mathcal{A}^{\star}_{\alpha,\mathcal{R}}) = \mathcal{R}^{\star}(\mathcal{L}(\mathcal{A}))$$

Proof 9 *Left-linearity, right-linearity and coherence of Theorem 3 are trivially satisfied.*

Lemma 3 *(Termination of tree automata completion) If $\mathcal{R}an(\alpha)$ is finite then completion terminates and the tree automaton $\mathcal{A}^{\star}_{\alpha,\mathcal{R}}$ is finite.*

Proof 10 *If $\mathcal{R}an(\alpha)$ is finite then the number of new states introduced by completion is finite. If the number of new states is finite then so is the number of states of $\mathcal{A}^{\star}_{\alpha,\mathcal{R}}$. Since one can only build a finite number of transitions on a finite set of states (and a finite alphabet $\mathcal{F}$), the set of transitions in $\mathcal{A}^{\star}_{\alpha,\mathcal{R}}$ is finite.*

Note that for injective abstraction function, proving that the range is finite is equivalent to proving that the domain is, i.e. that completion produce a finite number of distinct configurations to be normalized.

Now we give alternative algorithms and proofs of regularity of $\mathcal{R}^{\star}(E)$ for the classes described in section 2. For a regular language E and

- $\mathcal{R}$ **ground** [7, 3]: we use corollary 2 (ground TRSs are linear) and an injective abstraction α with a finite domain $\{r|_p \mid l \to r \in \mathcal{R} \text{ and } p \in \mathcal{P}os(r) \setminus \{\epsilon\}\}$. We can restrict α to this finite domain since in every new transition $f(t_1, \ldots, t_n) \to q$ added by the completion, $f(t_1, \ldots, t_n)$ is necessarily ground and is a right-hand side of a rule of $\mathcal{R}$. So it is enough to normalize $t_1, \ldots, t_n$ and all their subterms to normalize the transition. Hence, in α for every rule $l \to r$, every strict subterm of r is mapped to a new state. Since the domain is finite, so is the range and completion terminates.

- $\mathcal{R}$ **right-linear and monadic** [28]: we use theorem 3 and an abstraction function α with an empty domain which trivially satisfy the injectivity and coherence property w.r.t. $\mathcal{R}$ and $\mathcal{A}$. The domain of α is empty since every new transition produced by the completion is of the form $f(q_1, \ldots, q_n) \to q$ where $q_1, \ldots, q_n$ are states and do not need to be normalized. Assume that after each completion step, we determinize the completed automaton $\mathcal{A}^{n}_{\alpha,\mathcal{R}}$. Since the domain of α is empty, completion ends on $\mathcal{A}^{\star}_{\alpha,\mathcal{R}}$ which is determinized. Thanks to determinisation of the last completion step left-linearity condition is trivially satisfied and since the TRS is right linear, this is also the case for right-linearity condition.

- $\mathcal{R}$ **linear and semi-monadic** [6]: as in the ground case we define α as an injective function on the finite domain: $\{r|_p \mid l \to r \in \mathcal{R} \text{ and } p \in \mathcal{P}os_{\mathcal{F}}(r) \setminus \{\epsilon\}\}$. Similarly, we can restrict to this finite domain since in every new transition $f(t_1, \ldots, t_n) \to q$ added by the completion, t_i is either a ground term (and can be normalized by a single state) or is itself a state and thus does not need normalization.

$\mathcal{R}$ linear and "decreasing" [18]: Recall that "decreasing" means that every right-hand side is either a variable, or a term $f(t_1, \ldots, t_n)$ where $f \in \mathcal{F}$, $ar(f) = n$, and $\forall i = 1, \ldots, n$, t_i is a variable, a ground term, or a term whose variables do not occur in the left-hand side. For this class, the proof and abstraction function is similar to the linear and semi-monadic case except for variables occurring in the right-hand side but not in the left-hand side. For those variables, it is enough to substitute them by a specific state $q_{\mathcal{T}(\mathcal{F})}$ (which recognize $\mathcal{T}(\mathcal{F})$) and add the set of transitions $\{f(q_{\mathcal{T}(\mathcal{F})}, \ldots, q_{\mathcal{T}(\mathcal{F})}) \to q_{\mathcal{T}(\mathcal{F})} \mid f \in \mathcal{F}, ar(f) = n\}$ to the transitions of $\mathcal{A}$. As in the linear and semi-monadic case, we define α as an injective function on the finite domain: $\{r|_p \mid l \to r \text{ and } p \in \mathcal{P}os_{\mathcal{F}}(r) \setminus \{\epsilon\}\}$ where variables in r not occurring in l are substituted by $q_{\mathcal{T}(\mathcal{F})}$.

$\mathcal{R}$ constructor based [27]: in this particular case, there is also a restriction on the initial language $E = \{t\sigma\}$ where $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ is linear and $\sigma : \mathcal{X} \mapsto \mathcal{T}(\mathcal{C})$. Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be the tree automaton recognizing E. In this particular case, our aim is more to give an alternative algorithm rather than a proof of regularity. As in [27], we focus on the algorithm for left and right-linear TRSs since the left-linearity restriction can be discarded using determinization of tree automata (as in the right-linear and monadic case). We use theorem 3 ($\mathcal{R}$ is linear) and an injective abstraction function α such that $\mathcal{R}an(\alpha) \cap \mathcal{Q} = \emptyset$, thus left-linearity, right-linearity and coherence are satisfied. Now, let us prove that the domain of α is finite. Let $Q_{t\sigma}$ be the finite set of states necessary to normalize deterministically $t\sigma$, Q_{arg} be the set of states necessary to normalize the ground subterms of the right-hand sides of the rules and Δ_{arg} the related set of transitions. In [27], it is shown that for every defined symbol of t, for every substitution $\delta : \mathcal{X} \mapsto \mathcal{T}(\mathcal{F})$, for every rewriting $l\delta \to_{\mathcal{R}} r\delta$, there exists a substitution $\sigma : \mathcal{X} \mapsto Q_{t\sigma} \cup Q_{arg}$ such that $l\delta \to_{\Delta \cup \Delta_{arg}} l\sigma$ and $r\delta \to_{\Delta \cup \Delta_{arg}} r\sigma$. Hence, every critical pair encountered during the completion is of the form: $l\sigma \to_{\mathcal{A}} q$ and $l\sigma \to_{\mathcal{R}} r\sigma$ with $\sigma : \mathcal{X} \mapsto Q_{t\sigma} \cup Q_{arg}$. Since the number of defined symbols of t is finite, since $Q_{t\sigma} \cup Q_{arg}$ is finite, then so is the set of every possible critical pair and so is the domain of α.

In this last case, we did not give an explicit definition of α. The good news, and this is one of the main interest of tree automata completion algorithm in practice, is that it is useless to define α since it can be constructed automatically *during* completion. For all the above classes, since the domain of α is finite and since α is injective, we can construct an injective α function "on-the-fly" by associating a new state to every new configuration to normalize during completion. This lead to a fully automatic and terminating completion algorithm covering all the decidable classes we summed up here.

For building α on the fly, it is enough to start a completion with an empty abstraction function α and to create a new state $q \notin \mathcal{R}an(\alpha)$ and a new association $c \mapsto q$ in α for every new configuration c to normalize. If completion terminates (and it is necessarily the case for all decidable decidable classes we saw) then the completed automaton $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ recognizes $\mathcal{R}^\star(E)$ if $\mathcal{R}$ is linear (or if $\mathcal{R}$ is right-linear and $\mathcal{R}$ and $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ satisfy the left-linearity condition). Note that this algorithm even covers some decidable cases that are not

included in the above decidable classes. A very simple example is TRS $\mathcal{R} = \{f(s(x)) \to g(s(x)), g(s(x)) \to h(s(x))\}$ and initial language $E = \{f(s^\star(a))\}$. The set $\mathcal{R}^\star(E)$ is clearly regular but this example is outside of the decidable classes we saw. However, this TRS is linear and completion terminates with an injective abstraction function built on the fly so we have a proof of regularity of $\mathcal{R}^\star(E)$ and it is recognized by the completed automaton $\mathcal{A}^\star_{\alpha,\mathcal{R}}$.

Here is a first example showing how this result can be used with the Timbuk tool implementing usual operations on the tree automata and the completion algorithm.

Example 6 *Let us consider the following example given in* Timbuk *syntax:*

```
Ops 0:0 s:1 plus:2 even:1 odd:1 true:0 false:0
Vars x y z
TRS R
    plus(0, x) -> x
    plus(s(x), y)-> s(plus(x, y))
    even(0) -> true
    even(s(0)) -> false
    even(s(x)) -> odd(x)
    odd(0) -> false
    odd(s(0)) -> true
    odd(s(x)) -> even(x)

Automaton A
States qf qodd qeven qpo qpe
Final States qf
Transitions
        even(qpo) -> qf                 even(qpe) -> qf
        plus(qodd, qodd) -> qpo         s(qeven) -> qodd        s(qodd) -> qeven
        plus(qeven, qeven) -> qpe       0 -> qeven

Automaton Reach
States qf
Final States qf
Transitions
        false -> qf
```

Where $\mathcal{R}$ defines the 'plus' function and the 'even' and 'odd' predicates on the naturals, and the tree automaton $\mathcal{A}$ defines the language $E = \{even(plus(t_1, t_2))\}$ where t_1, t_2 are either two even or two odd naturals. This example is in Réty's class. The language $\mathcal{R}^\star(E)$ is regular and can be automatically computed by Timbuk *within some milliseconds:*

```
Automaton current
States qnew3:0 qnew2:0 qnew1:0 qnew0:0 qf:0 qodd:0 qeven:0 qpo:0 qpe:0
Final States qf

Prior
  plus(qodd,qodd) -> qnew3      plus(qeven,qeven) -> qnew2
  plus(qodd,qeven) -> qnew0     plus(qeven,qodd) -> qnew1

Transitions
  even(qpo) -> qf               even(qpe) -> qf         plus(qodd,qodd) -> qpo
  s(qeven) -> qodd              s(qodd) -> qeven        plus(qeven,qeven) -> qpe
  0 -> qeven                    s(qnew1) -> qpo         plus(qeven,qodd) -> qnew1
  s(qodd) -> qpe                0 -> qpe                s(qnew0) -> qpe
  plus(qodd,qeven) -> qnew0     s(qeven) -> qnew1       s(qnew3) -> qnew1
  plus(qodd,qodd) -> qnew3      s(qnew2) -> qnew0       plus(qeven,qeven) -> qnew2
  true -> qf                    odd(qnew1) -> qf        odd(qodd) -> qf
  odd(qnew0) -> qf              s(qnew1) -> qnew3       0 -> qnew2
```

```
  s(qodd) -> qnew2              s(qnew0) -> qnew2       even(qeven) -> qf
  even(qnew2) -> qf             even(qnew3) -> qf
```

Note that the specific subset of transitions denoted by `Prior` *represents in this case the injective abstraction function α that has been built automatically. If we compute the intersection between $\mathcal{R}^\star(E)$ and the Reach automaton recognizing the term true, we obtain an empty automaton:*

```
Intersection with Reach gives (the empty automaton):

States
Final States

Transitions
```

which means that $false$ is not reachable. Thus we have proved that $even(plus(t_1, t_2))$ where t_1 and t_2 are either both even or both odd numbers cannot rewrite to $false$. However, we are still not sure that it always rewrites to $true$. This can be done in the following way: Timbuk *can compute the tree automaton recognizing $IRR(\mathcal{R})$ which is:*

```
Automaton Nf
States q3:0 q2:0 q1:0 q0:0
Final States q0 q1 q2 q3

Transitions
  false -> q3           true -> q3              odd(q3) -> q3
  even(q3) -> q3        plus(q3,q3) -> q3       0 -> q2
  plus(q3,q2) -> q3     s(q3) -> q1             s(q2) -> q0
  s(q0) -> q1           s(q1) -> q1             plus(q3,q0) -> q3
  plus(q3,q1) -> q3
```

and the intersection between $\mathcal{R}^\star(E)$ and $IRR(\mathcal{R})$ gives $\mathcal{R}^!(E)$:

```
Intersection with Nf gives (not empty):
States q0:0
Final States q0
Transitions
  true -> q0
```

which means that $\mathcal{R}^!(E) = \{true\}$. Hence, every term of E necessarily rewrites to $true$[3].

5 From exactness to approximation

One of the main interest of this algorithm is the ability to switch on the fly from exact to approximate computations. Using approximation may be necessary in several contexts. When the construction of $\mathcal{R}^\star(E)$ does not converge (because it is not regular) approximations force completion to terminate on an automaton over-approximating $\mathcal{R}^\star(E)$. When the completion is too long ($\mathcal{R}^\star(E)$ is regular but too big) approximations permit to accelerate completion. In general completion diverges because it produces an infinite set of transitions used to recognize an infinite set of new distinct reachable terms. The idea behind approximation is to explicitly merge together (or to identify) some terms in order to limit the set of new transitions necessary to recognize them.

[3] Since $\mathcal{R}$ is terminating, every term of E has a normal form

Merging terms together is similar to defining an equivalence class on terms. In order to merge together two terms s and t in a common equivalence class, an usual way is to use an equation $s = t$. For instance, in order to build an approximation where terms 0 and $s(0)$ are equivalent, it is enough to achieve a completion with an additional *approximation equation* $0 = s(0)$[4]. Similarly, to build an approximation where every natural number is abstracted by its parity (even or odd) it is enough to perform a completion with an additional approximation equation $s(s(x)) = x$.

In Timbuk, for applying an approximation equation $l = r$ to an automaton $\mathcal{A}$, we simply search for $\mathcal{Q}$-substitutions $\sigma : \mathcal{X} \mapsto \mathcal{Q}$ and states $q \in \mathcal{Q}$ such that $l\sigma \rightarrow_{\mathcal{A}}^* q$. Then, for every state q' different from q and such that $r\sigma \rightarrow_{\mathcal{A}}^* q'$, we merge together q and q', i.e. every occurrence of state q in $\mathcal{A}$ is renamed by q' (or q' is renamed by q since the recognized language will be the same). Now, let us give a small example of what can be done using such approximations.

Example 7 *Let us consider the following* Timbuk *specification:*

```
Ops 0:0 s:1 plus:2 times:2 square:1 true:0 false:0 even:1 odd:1
Vars x y z
TRS R
    plus(0, x) -> x
    plus(s(x), y)-> s(plus(x, y))
    times(0, x) -> 0
    times(s(x), y) -> plus(y, times(x, y))
    square(x) -> times(x, x)

    even(0) -> true
    even(s(0)) -> false
    even(s(x)) -> odd(x)
    odd(0) -> false
    odd(s(0)) -> true
    odd(s(x)) -> even(x)

    even(square(x)) -> odd(square(s(x)))
    odd(square(x)) -> even(square(s(x)))

Automaton A0
States q0 q1 q2 q3 qf1
Final States qf1
Transitions
        0 -> q0         square(q0) -> q2        even(q2) -> qf1

Automaton Reach
States q0
Final States q0
Transitions
        false -> q0
```

where $E = \{even(square(O))\}$. *Note that* $even(square(O))$ *may be rewritten by* $\mathcal{R}$ *into* $odd(square(s(O)))$ *and into* $even(square(s(s(O))))$ *and so on. Moreover each of these terms may be rewritten by the definition of 'square', 'even' and 'odd'. If we try to build the abstraction function* α *on the fly using new states, the completion diverges. After the* 9*-th step of completion,* $\mathcal{A}^9_{\alpha,\mathcal{R}}$ *has* 201 *transitions and completion is still not over. In order to force completion to terminate, it is possible to add an approximation equation:*

[4]This in fact will have an even stronger effect since it will collapse together all the natural numbers.

```
Type additional equations and end by a dot '.':

s(s(x))=x.
```

and this merges together 23 *states of* $\mathcal{A}^{9}_{\alpha,\mathcal{R}}$ *and it now have only* 104 *transitions. Then completion continues and approximation equation is applied after every step until we reach step* 12 *where* $\mathcal{A}^{12}_{\alpha,\mathcal{R}} = \mathcal{A}^{13}_{\alpha,\mathcal{R}}$ *and thus* $\mathcal{A}^{12}_{\alpha,\mathcal{R}} = \mathcal{A}^{\star}_{\alpha,\mathcal{R}}$. *This last automaton has only* 25 *transitions but it is complete w.r.t.* $\mathcal{R}$. *Then if we compute the intersection between* $\mathcal{A}^{\star}_{\alpha,\mathcal{R}}$ *and the automaton Reach recognizing only the term* $'false'$ *we obtain an empty automaton. Thus, we have proved that for all natural number* n, *if* n *is even (resp. odd) then so is* n^2.

Timbuk also provide another tool to describe approximations: *approximation rules* which are more related to automata structure and thus more precise. Approximation rules are necessary when approximation equations are not expressive or precise enough to build adequate approximations. Approximation rules also offer more control on the domain and the range of the abstraction function α and thus let the user ensure termination of the completion anytime he needs to.

The general form for approximation rules is the following: $[s \to x] \to [l_1 \to x_1, \ldots, l_n \to x_n]$ where $[s \to x]$ with $s \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}, \mathcal{X})$ and $x \in \mathcal{X} \cup \mathcal{Q}$ is a pattern to be matched over the new transitions $t \to q'$ obtained by completion and $[l_1 \to x_1, \ldots, l_n \to x_n]$ are rules used to normalize t. The syntactical constraint for those rules is the following: $l_i \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q}, \mathcal{X})$ and either $x_i \in \mathcal{Q}$ or $x_i \in \mathcal{V}ar(l_i) \cup \mathcal{V}ar(s) \cup \{x\}$. To normalize a transition of the form $t \to q'$, we match s on t and x on q', obtain a given substitution σ and then we normalize t with the rewrite system $\{l_1\sigma \to r_1\sigma, \ldots, l_n\sigma \to r_n\sigma\}$ where $r_1\sigma, \ldots, r_n\sigma$ are necessarily states. For example, normalizing a transition $f(h(q_1), g(q_2)) \to q_3$ with approximation rule $[f(x, g(y)) \to z] \to [g(u) \to z]$ will give a substitution $\sigma = \{x \mapsto h(q_1), y \mapsto q_2, z \mapsto q_3\}$, an instantiated set of rewrite rules $[g(u) \to q_3]$. Thus, $f(h(q_1), g(q_2)) \to q_3$ will be normalized into a normalized transition $g(q_2) \to q_3$ and a partially normalized transition $f(h(q_1), q_3) \to q_3$. Some examples of the use of approximation rules in practice are given in section 6.3.

6 Application examples

In this section, we present three examples of application for $\mathcal{R}^{\star}(E)$ and $\mathcal{R}^{!}(E)$.

6.1 Sufficient completeness

This property has already been much investigated [4, 21, 23, 19], in the context of algebraic specifications. We give here a definition of sufficient completeness of a TRS on a subset of the set of ground terms $E \subseteq \mathcal{T}(\mathcal{F})$.

Definition 11 *A TRS* $\mathcal{R}$ *is sufficiently complete on* $E \subseteq \mathcal{T}(\mathcal{F})$ *if* $\forall s \in E$, $\exists t \in \mathcal{T}(\mathcal{C})$ *s.t.* $s \to^{*}_{\mathcal{R}} t$, *where* $\mathcal{C}$ *is the set of constructors in* $\mathcal{F}$.

Usual methods for checking this property on algebraic specifications are either based on enumeration and testing techniques [21, 23, 19] or on disunification [4]. We propose, here, to check this property thanks to the set $\mathcal{R}^!(E)$.

Proposition 3 *If the TRS $\mathcal{R}$ is weakly normalizing on $E \subseteq \mathcal{T}(\mathcal{F})$, and $\mathcal{R}^!(E) \subseteq \mathcal{T}(\mathcal{C})$, then $\mathcal{R}$ is sufficiently complete on E.*

This comes from the fact that since $\mathcal{R}$ is weakly normalizing on E, for all terms $s \in E$, $\exists t \in IRR(\mathcal{R})$ s.t. $s \to_{\mathcal{R}}^* t$. Moreover, $t \in \mathcal{R}^!(E)$. Since $\mathcal{R}^!(E) \subseteq \mathcal{T}(\mathcal{C})$, we have $t \in \mathcal{T}(\mathcal{C})$.

Example 8 *Looking back to the example 6, one can remark that $\mathcal{R}^!(E) = \{true\} \subseteq \mathcal{T}(\mathcal{C})$ hence, the TRS $\mathcal{R}$ is sufficiently complete on E.*

On the other hand, sufficient completeness on E does not necessarily imply that $\mathcal{R}^!(E) \subseteq \mathcal{T}(\mathcal{C})$. For example, let $\mathcal{R} = \{f(a) \to a, f(a) \to f(b)\}$, $\mathcal{C} = \{a, b\}$ and let $E = \{f(a)\}$. Then $\mathcal{R}$ is sufficiently complete on E, since $f(a) \to_{\mathcal{R}} a$, but $\mathcal{R}^!(E) = \{a, f(b)\} \not\subseteq \mathcal{T}(\mathcal{C})$. Note that using tree automata permits to give a very precise description of the domain on which a function is complete, more precise that what can be done with simple types for instance.

6.2 Strong non-termination

Definition 12 *(Strong non-termination) Let E be a set of terms and $\mathcal{R}$ be a TRS. The TRS $\mathcal{R}$ is said to be* strongly non-terminating *if there exists no finite $\mathcal{R}$-rewrite chains from terms of E.*

Theorem 4 *A TRS $\mathcal{R}$ is non-terminating on E is $\mathcal{R}^!(E) = \emptyset$.*

Proof 11 *Obvious, since $\mathcal{R}^!(E) = \emptyset$ means that every term of E is reducible, and so are every terms $\mathcal{R}$-reachable from E.*

When the TRS represents some parallel processes, the non-termination property is close to the deadlock-free property. Let us show a very simple example of this aspect.

Example 9 *Assume that we have two processes each one having a list of elements to count. Assume that the counter is a shared variable that should not be accessed by the two processes at the same time. Each process has two possible states 'busy' if it accesses to the shared counter or 'free' otherwise. A similar flag is associated to the shared counter in order to protect it from a concurrent access. The behavior of this system is described by the following TRS $\mathcal{R}$ where x, y, z, u are variables, Proc represents a process, cons and null are used to build the lists and S represents a configuration of the system:*

$$
\begin{aligned}
&S(Proc(free, cons(x, y)), z, free, u) \to S(Proc(busy, cons(x, y)), z, busy, u)\\
&S(Proc(busy, cons(x, y)), z, busy, u) \to S(Proc(free, y), z, free, s(u))\\
&S(z, Proc(free, cons(x, y)), free, u) \to S(z, Proc(busy, cons(x, y)), busy, u)\\
&S(z, Proc(busy, cons(x, y)), busy, u) \to S(z, Proc(free, y), free, s(u))\\
&S(Proc(x, null), Proc(y, null), z, u) \to S(Proc(x, null), Proc(y, null), z, u)
\end{aligned}
$$

The initial language E is recognized by the following tree automaton $\mathcal{A}$ whose final state is q_0 and set of transitions is:

$$\begin{array}{lll} S(q_1,q_1,q_2,q_3) \to q_0 & free \to q_2 & 0 \to q_3 \\ Proc(q_2,q_4) \to q_1 & null \to q_4 & cons(q_3,q_4) \to q_4 \end{array}$$

The set E contains terms of the form $S(Proc(free,l_1),Proc(free,l_2),free,0)$ where l_1 and l_2 are lists of 0. Using an exact abstraction function α the completion does not terminate. On the 7-th completion step the completed tree automaton $\mathcal{A}^7_{\alpha,\mathcal{R}}$ contains 41 transitions. In order to make the completion terminate, we choose to add an approximation equation $s(x) = x$ which merge some states and transition of $\mathcal{A}^7_{\alpha,\mathcal{R}}$ together so that the tree automaton contains only 19 transitions. Finally $\mathcal{A}^8_{\alpha,\mathcal{R}} = \mathcal{A}^7_{\alpha,\mathcal{R}}$, hence $\mathcal{A}^\star_{\alpha,\mathcal{R}} = \mathcal{A}^7_{\alpha,\mathcal{R}}$. The automaton $\mathcal{A}^\star_{\alpha,\mathcal{R}}$ over approximating $\mathcal{R}^\star(E)$ is the following:

```
Ops S:4 Proc:2 cons:2 null:0 busy:0 free:0 s:1 o:0
Automaton current
States qnew10:0 qnew9:0 qnew8:0 qnew7:0 qnew6:0 qnew5:0 qnew4:0
       qnew3:0 qnew2:0 qnew1:0 qnew0:0 q0:0 q1:0 q2:0 q3:0 q4:0
Final States q0
Prior
  s(qnew10) -> qnew10          Proc(qnew5,q4) -> qnew4        free -> qnew5
  Proc(qnew1,qnew3) -> qnew0   cons(qnew10,q4) -> qnew3       busy -> qnew1

Transitions
  S(q1,q1,q2,qnew10) -> q0        free -> q2
  o -> qnew10                     Proc(q2,q4) -> q1
  null -> q4                      cons(qnew10,q4) -> q4
  busy -> qnew1                   cons(qnew10,q4) -> qnew3
  Proc(qnew1,qnew3) -> qnew0      free -> qnew5
  Proc(qnew5,q4) -> qnew4         S(qnew0,qnew4,qnew1,qnew10) -> q0
  S(qnew0,q1,qnew1,qnew10) -> q0  S(qnew4,qnew0,qnew1,qnew10) -> q0
  S(q1,qnew0,qnew1,qnew10) -> q0  S(qnew4,q1,qnew5,qnew10) -> q0
  S(q1,qnew4,qnew5,qnew10) -> q0  S(qnew4,qnew4,qnew5,qnew10) -> q0
  s(qnew10) -> qnew10
```

Now, if we compute the intersection with $IRR(\mathcal{R})$ we obtain an automaton over-approximating $\mathcal{R}^!(E)$. The automaton obtained by intersection recognizes an empty language. Hence, we also have $\mathcal{R}^!(E) = \emptyset$ and thus $\mathcal{R}$ is strongly non-terminating on E.

6.3 Reachability testing

In this part, we focus on the applications of negative reachability testing, i.e. using over-approximations of $\mathcal{R}^\star(E)$ to show that $s \not\to_{\mathcal{R}}{}^* t$. The positive result (i.e. the exact cases for $\mathcal{R}^\star(E)$ which permits to show properties of the form $s \to_{\mathcal{R}}{}^* t$) of section 4 is more recent but it should quickly find some applications in theorem proving on equational theories and in verification.

Several experiments have been done on negative reachability testing for proving properties over functional programs [11], communicating parallel processes [15], but the most significant experiment has been done on cryptographic protocol. First, Timbuk has been used to prove secrecy and authentication properties on the Needham-Schroder Public Key Protocol which is a typical case study for verification methods [12]. More recently it was used to prove an anti-replay property on a protocol of the SmartRight system designed by Thomson Multimedia for digital rights management [13].

In this setting, the used TRSs are highly non-terminating and user defined approximation reveals to be a very powerful and flexible way to over-approximate the set of reachable terms. In those works, using over-approximations have led to semi-automatic proof of some properties that require induction, lemmas and user expertise when they are proved in a proof assistant.

Let us show some particular aspects of the TRSs and approximation rules used for verifying cryptographic protocols. Those protocols are supposed to be secure in an hostile environment where an intruder stores every message and every key he sees, decrypts some parts, forges new messages with the parts he has and sends every possible message in its store in order to attack some agents. We can model the intruder store using a term built with an Associative Commutative (AC) symbol `store`, where for example the term `store(a, store(store(b, a), c))` represents the multiset $\{\!\{a, a, b, c\}\!\}$. The terms `pubkey(x)`, `privkey(x)`, `encr(k, c)`, and `cons(x, y)` represent respectively the public and private key of an agent `x`, the encryption of `c` using the key `k` and a message composed of two parts `x` and `y`. We can model some of the message constructions that an intruder can do on its store, like it is done for example in [26]:

```
(* The intruder can encrypt any stored component with any stored key *)
store(z, pubkey(x)) -> store(encr(pubkey(x), z), store(z, pubkey(x)))
store(z, privkey(x)) -> store(encr(privkey(x), z), store(z, privkey(x)))

(* The intruder can decompose or compose any component he has *)
store(cons(x,y), m) -> store(store(cons(x,y), m),store(x, y))
store(x, y) -> store(cons(x, y), store(x,y))

(* The intruder can decrypt a message if he has the related key *)
store(encr(pubkey(x), z), privkey(x)) ->
        store(encr(pubkey(x), z), store(privkey(x), z))
store(encr(privkey(x), z), pubkey(x)) ->
        store(encr(privkey(x), z), store(pubkey(x), z))
```

The rules encoding the AC behavior of the `store` symbol are also necessary:

```
store(x, y) -> store(y, x)
store(store(x, y), z) -> store(x, store(y, z))
store(x, store(y, z)) -> store(store(x, y), z)
```

There are two particular things to remark on those rules. First, they are all non terminating, we thus have to define strong approximation rules in order to restrain divergence of the completion. Second, rules for decryption are non left-linear and we must check the left-linearity condition on the TRS and the completed tree automaton.

Now, let us show some of the approximation rules we use in this particular case. When AC symbols are simply used for representing sets of objects, a quite natural approximation rule

for the `store` symbol is the following: `[store(x, y) -> z] -> [x -> z  y -> z]`. This rule normalizes every new configuration of the form `store(s, t) -> q` (where `s` and `t` are not states) into configurations `s -> q`, `t -> q` and `store(q, q) -> q`. The intuition behind this rule is that every 'subset' `x` and `y` of the store `store(x, y)` should be recognized by the same state as `store(x, y)`. Similarly the approximation rule dealing with the decryption rules is of the form: `[encr(pubkey(qA), y) -> z] -> [y -> qAsecret]` where `qA` is the state recognizing the agent `A` and `qAsecret` is a state used to recognize the language of terms protected by the public key of `A` and thus that should remain secret during the protocol execution.

Those simple approximation rules permit to restrain the divergence of rewriting into a finite (and approximated) set of reachable terms recognized by a finite tree automaton. Then, what remains to be proved is that the completed automaton and the TRS fulfills the left-linearity condition. This can easily and automatically be checked using the simple left-linearity condition (see definition 8). During completion, ensuring this property is easy since every non left-linear variables of the TRS match agent names (like in the above rules). Hence, it is enough to build an approximation such that agent names are deterministically recognized[5] in order to ensure the simple left-linearity condition.

7 Implementing tree automata completion

In this section, we briefly present the Timbuk tool [14] in which the tree automata completion is implemented. We also discuss the efficiency of the matching algorithm over tree automata used to find critical pairs during completion. Timbuk is a tree automata library providing basic primitives on non deterministic tree automata like intersection, union, complement of languages, determinisation of tree automata, construction of $IRR(\mathcal{R})$ for left-linear TRS, as well as the tree automata completion algorithm with some tools for building abstraction function by hand or automatically. The current distributed Timbuk 1.1 library is written in Ocaml [22], contains nearly no specific optimisation, can only build over-approximations and provide only approximation rules (see section 5) to construct abstraction functions by hand. Now, we are finishing version 2.0 which includes some improvements on the matching algorithm described in the next sections, some new automatic normalization strategies (in particular an exact one corresponding to the results of section 4) and the approximation equation facility described in section 5. Timbuk 2.0 is still written in Ocaml and will soon be available.

Let us now present the basic matching algorithm and the optimised one. Recall that the matching problem, for a given rewriting rule $l \to r$ and a tree automaton $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, consists in computing all the $\mathcal{Q}$-substitutions σ such that there is a state $q \in \mathcal{Q}$ and $l\sigma \to_\Delta^\star q$.

[5] For instance by fixing $\alpha(X) = q_X$ for every agent X.

7.1 Basic matching algorithm

This algorithm proposed in [16] is close to a standard matching algorithm on terms. It is defined using deduction rules over specific formulas called *matching problems*. In the following, a *matching problem* is a quantifier-free first order formula build on literals $\bot$, $s \trianglelefteq c$ where $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, and closed by the connectives $\vee$ and $\wedge$. An empty conjunction $\bigwedge_\emptyset$ is a trivially true matching problem.

Definition 13 *Let ϕ, ϕ_1, ϕ_2 be matching problems, $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ be a term, $c \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, and $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ a tree automaton. A* solution *to the matching problem ϕ is a $\mathcal{Q}$-substitution $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ such that*

- *if $\phi = s \trianglelefteq c$, then $s\sigma \to_\Delta^* c$, or*
- *if $\phi = \phi_1 \wedge \phi_2$, then σ is a solution of ϕ_1 and a solution of ϕ_2, or*
- *if $\phi = \phi_1 \vee \phi_2$, then σ is a solution of ϕ_1 or a solution of ϕ_2.*

We assume that matching is applied on automata without epsilon-transitions. An epsilon transition is a transition of the form $q \to q'$ where q and q' are states. Any set of transition $\Delta \cup \{q \to q'\}$ can be equivalently replaced by $\Delta \cup \{c \to q' \mid c \to q \in \Delta\}$. Now let us give the matching algorithm.

Definition 14 *Let $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be a tree automaton, $f \in \mathcal{F}$, $ar(f) = n$, $g \in \mathcal{F}$, $ar(g) = m$, $q, q_1, \ldots, q_n \in \mathcal{Q}$, $q'_1, \ldots, q'_n \in \mathcal{Q}$, $c_1, \ldots, c_d \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$, $s, s_1, \ldots, s_n \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and ϕ_1, ϕ_2, ϕ_3 be non-empty matching problems. The matching algorithm consists in normalizing any matching problem of the form $s \trianglelefteq q$ by the following set of rules.*

Decompose
$$\frac{f(s_1, \ldots, s_n) \trianglelefteq f(q_1, \ldots, q_n)}{s_1 \trianglelefteq q_1 \wedge \ldots \wedge s_n \trianglelefteq q_n}$$

Clash
$$\frac{f(s_1, \ldots, s_n) \trianglelefteq g(q'_1, \ldots, q'_m)}{\bot}$$

Configuration
$$\frac{s \trianglelefteq q}{s \trianglelefteq c_1 \vee \ldots \vee s \trianglelefteq c_d \vee \bot}$$

if $s \notin \mathcal{X}$, for all $c_i \in \mathcal{T}(\mathcal{F} \cup \mathcal{Q})$ $i = 1 \ldots d$ such that $c_i \to q \in \Delta$.

Moreover, after each application of any of these rules, matching problems are normalized by the following set of rules ξ:

$$\frac{\phi_1 \wedge (\phi_2 \vee \phi_3)}{(\phi_1 \wedge \phi_2) \vee (\phi_1 \wedge \phi_3)} \qquad \frac{\phi_1 \vee \bot}{\phi_1} \qquad \frac{\phi_1 \wedge \bot}{\bot}$$

Correction, completeness and termination of the algorithm comes from the following theorem of [16].

Theorem 5 *Given $s \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, and $q \in \mathcal{Q}$, every matching problem $s \trianglelefteq q$ has a normal form such that*

- *if it is $\bot$ then there is no $\mathcal{Q}$-substitution σ s.t. $s\sigma \rightarrow^*_{\Delta'} q$,*
- *if it is empty, then for all $\mathcal{Q}$-substitution σ, $s\sigma \rightarrow^*_{\Delta'} q$,*
- *otherwise, the normal form is a disjunction $\bigvee_{i=1}^{k} \phi_i$ s.t. $\phi_i = \bigwedge_{j=1}^{n_i} x_j^i \trianglelefteq q_j^i$, where $x_j^i \in \mathcal{X}$ and $q_j^i \in \mathcal{Q}$, and $\sigma_1 = \{x_j^1 \mapsto q_j^1 \mid j = 1 \ldots n_1\}, \ldots, \sigma_k = \{x_j^k \mapsto q_j^k \mid j = 1 \ldots n_k\}$ are the only $\mathcal{Q}$-substitutions s.t. $s\sigma_i \rightarrow^*_{\Delta'} q$.*

Thanks to this algorithm, for a given rule $l \rightarrow r$ and a given state q, it is possible to find every $\mathcal{Q}$-substitution σ s.t. $l\sigma \rightarrow^*_{\Delta} q$.

Example 10 *Let $A = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$, where $\mathcal{F} = \{f, g, a\}$, $\mathcal{Q} = \{q_0, q_1\}$, $\mathcal{Q}_f = \{q_0\}$ and $\Delta = \{f(q_1) \rightarrow q_0, g(q_1) \rightarrow q_1, a \rightarrow q_1\}$. The language $\mathcal{L}(A) = \{f(g^*(a))\}$. Let $\mathcal{R} = \{f(g(x)) \rightarrow g(f(x))\}$. If we apply matching on $f(g(x)) \trianglelefteq q_0$, we obtain the following deductions, where the name of the applied rule is given on the right, and normalization with simplification rules are omitted:*

$f(g(x)) \trianglelefteq q_0$	*rule* **Configuration**
$f(g(x)) \trianglelefteq f(q_1)$	*rule* **Decompose**
$g(x) \trianglelefteq q_1$	*rule* **Configuration**
$g(x) \trianglelefteq g(q_1) \ \vee \ g(x) \trianglelefteq a$	*rule* **Clash**
$g(x) \trianglelefteq g(q_1)$	*rule* **Decompose**
$x \trianglelefteq q_1$	

Let σ be the $\mathcal{Q}$-substitution $\sigma = \{x \mapsto q_1\}$. Thus, we deduced that $l\sigma = f(g(q_1)) \rightarrow^*_{\Delta} q_0$.

7.2 An optimised algorithm

We propose here a more efficient algorithm: we represent a rewriting system $\mathcal{R}$ with a tree automaton $\mathcal{A}_{\mathcal{R}}$, which will permit us to compute all the critical pairs between $\mathcal{R}$ and a regular language $\mathcal{A}$ thanks to $\mathcal{A} \cap \mathcal{A}_{\mathcal{R}}$. First, for every term $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, we define a tree automaton which language is exactly $\{t\}$ using abstraction and normalization functions defined in section 3.

Definition 15 *Term automaton Let $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, and consider $\mathcal{S}$ the set of all the subterms of t, $\mathcal{Q}_t$ a set of state, and $\alpha : \mathcal{S} \rightarrow \mathcal{Q}_t$ an injective abstraction function. The term automaton for t is defined by $\mathcal{A}_{\alpha,t} = \langle \mathcal{F}, \mathcal{Q}_t, \mathcal{Q}_{tf}, \Delta_t \rangle$ where $\mathcal{Q}_{ft} = \{top_\alpha(t)\}$ and $\Delta_t = Norm_\alpha(t \rightarrow top_\alpha(t))$.*

Proposition 4 *Consider $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, α an injective abstraction and $\mathcal{A}_{\alpha,t}$ its term automaton,*

$$\mathcal{L}(\mathcal{A}_{\alpha,t}) = \{t\}$$

Proof 12 *The proof is an inductive reasoning over the depth of t:*

- *If $depth(t) = 1$ then $\mathcal{A}_{\alpha,t} = \langle \mathcal{F}, \{top_\alpha(t)\}, \{top_\alpha(t)\}, \{t \to top_\alpha(t)\}\rangle$, and $\mathcal{L}(\mathcal{A}_{\alpha,t}) = \{t\}$*
- *Let us assume that for all terms t such that $depth(t) \leq n$ then $\mathcal{L}(\mathcal{A}_{\alpha,t}) = \{t\}$.*

  Now, if $depth(t) = n + 1$ and $\mathcal{A}_{\alpha,t} = \langle \mathcal{F}, \bigcup_{t_i \in \mathcal{S}} \{top_\alpha(t_i)\}, \{top_\alpha(t)\}, Norm_\alpha(t \to top_\alpha(t))\rangle$, necessarily t is of the form $f(t_1, \ldots, t_n)$. Thanks to induction hypothesis, all subterms t_i are recognized by states $top_\alpha(t_i)$ then $t = f(t_1, \ldots, t_n) \to_\Delta f(top_\alpha(t_1), \ldots, top_\alpha(t_n)) \to_\Delta top_\alpha(t)$, and $\mathcal{L}(\mathcal{A}_{\alpha,t}) = \{t\}$.

Let us now define the automaton $\mathcal{A}_\cap = \mathcal{A}_{\alpha,t} \cap \mathcal{A}$ which recognizes the solutions of the matching of t on $\mathcal{A}$.

Definition 16 *Substitution automaton Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\rangle$ be an automaton, α an injective abstraction function and $\mathcal{A}_{\alpha,t} = \langle \mathcal{F}, \mathcal{Q}_t, \mathcal{Q}_{ft}, \Delta_t\rangle$ the automaton of $t \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, we define $\mathcal{A}_\cap = \langle \mathcal{F}, \mathcal{Q}_\cap, \mathcal{Q}_{f,\cap}, \Delta_\cap\rangle$ the substitution automaton of t in $\mathcal{A}$ by:*

- $\mathcal{Q}_\cap = \mathcal{Q}_t \times \mathcal{Q}$
- $\mathcal{Q}_{f,\cap} = \{\alpha(t)\} \times \mathcal{Q}$
- $\Delta_\cap =$
  $\{f((q_1, q'_1), \ldots, (q_n, q'_n)) \to (q_{n+1}, q'_{n+1}) \mid q_i \in \mathcal{Q}_t,\ q'_i \in \mathcal{Q},$
  $\quad f(q_1, \ldots, q_n) \to q_{n+1} \in \Delta_t,\ f(q'_1, \ldots, q'_n) \to q'_{n+1} \in \Delta\}$
  $\cup\{(x, q) \to (q_x, q) \mid x \to q_x \in \Delta_t,\ q \in \mathcal{Q}\}$

The set of substitution solution for $\mathcal{A}_\cap$ is defined in the following way.

Definition 17 *Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta\rangle$ be an automaton, t a term, $\mathcal{A}_{t,\alpha}$ its automaton and $\mathcal{A}_\cap$ the substitution automaton of t in $\mathcal{A}$. Let $s \in \mathcal{T}(\mathcal{F}, \mathcal{X} \times \mathcal{Q})$, if s is recognized by a state of $\mathcal{A}_\cap$ then s defines a set of $\mathcal{Q}$-substitutions σ:*

- *if $s = (x, q)$, then s defines the $\mathcal{Q}$-substitution $\{x \mapsto q\}$*
- *if $s = (a, a)$ then s defines the empty substitution*
- *if $s = f(s_1, \ldots, s_n)$, then if $\{\sigma_{i,j}\}_{i \in I_j}$ are the substitutions associated to s_j, s define the set of substitutions $\tau = \sigma_{1,i_1} \circ \ldots \circ \sigma_{n,i_n}$.*

Example 11 *Let A_1 be the automaton for term $f(x, g(y))$. Let A_2 be the automaton recognizing the language $\{f(a, g^\star(b)), f(g^\star(b), a)$. Let A_3 be the intersection automaton ($A_\cap$). This last automaton recognize a unique term $f((qx, q1), g((qy, q3))$ which defines the $\mathcal{Q}$-substitution $\sigma = \{x \mapsto q_1,\ y \mapsto q_3\}$*

A_1 *with*
$\mathcal{Q} = \{qx, qy, qf, qg\}$
$\mathcal{Q}_f = \{qf\}$
$\Delta =$

$$\begin{array}{rcl} y & \to & qy \\ g(qy) & \to & qg \\ x & \to & qx \\ f(qx, qg) & \to & qf \end{array}$$

A_2 *with*
$\mathcal{Q} = \{q1, q2, q3, q4\}$
$\mathcal{Q}_f = \{q4\}$
$\Delta =$

$$\begin{array}{rcl} a & \to & q1 \\ b & \to & q2 \\ g(q2) & \to & q3 \\ g(q3) & \to & q3 \\ f(q1, q3) & \to & q4 \\ f(q3, q1) & \to & q4 \end{array}$$

A_3 *with*
$\mathcal{Q} = \{qx, qy, qf, qg\} \times \{q1, q2, q3, q4\}$
$\mathcal{Q}_f = \{qf\} \times \{q1, q2, q3, q4\}$
$\Delta =$

$$\begin{array}{rcl} (y, q2) & \to & (qy, q2) \\ (y, q3) & \to & (qy, q3) \\ (x, q1) & \to & (qx, q1) \\ (x, q3) & \to & (qx, q3) \\ g((qy, q2)) & \to & (qg, q2) \\ g((qy, q3)) & \to & (qg, q3) \\ f((qx, q1), (qg, q3)) & \to & (qf, q4) \\ f((qx, q3), (qg, q1)) & \to & (qf, q4) \end{array}$$

Theorem 6 *Let $\mathcal{A} = \langle \mathcal{F}, \mathcal{Q}, \mathcal{Q}_f, \Delta \rangle$ be an automaton, $l \to r$ a rewriting rule, and $\mathcal{A}_\cap = \mathcal{A} \cap \mathcal{A}_{\alpha,l}$, if $\mathcal{A}$ respects the left-linearity condition induced by $l \to r$ then the set of $\mathcal{Q}$-substitutions defined by all the states $(\alpha(l), q)$ in $\mathcal{A}_\cap$ is exactly the set of $\mathcal{Q}$-substitutions $\{\sigma_i\}_{i \in I}$ such that $l\sigma_i \to^\star_\Delta q$ in $\mathcal{A}$.*

Proof 13 *1. First assume that $\{\sigma_i\}$ is a set of $\mathcal{Q}$-substitutions defined by a term s such that $s \to^\star (\alpha(l), q)$ in $\mathcal{A}_\cap$ we prove that $l\sigma \to^\star q$ with an inductive reasoning over the derivation $s \to^n (\alpha(l), q)$:*

- *If $s \to (\alpha(l), q)$ then $depth(s) = 1$ and*
  - *either $s = (a, a)$ where a is a constant, we have $(a, a) \to_{\mathcal{A}_\cap} (\alpha(a), q)$ then $l = a$ and $l \to_\Delta q$, s defines the substitution of empty domain.*
  - *or $s = (x, a)$ we have also $(x, a) \to_{\mathcal{A}_\cap} (\alpha(a), q)$ then $l = x$, and s defines $\sigma = \{x \mapsto q\}$, we have $l\sigma = a$ and $a \to q$*
- *Assume now that for all set $\{\sigma_i\}_{i \in I}$ defined by a term s of $\mathcal{A}_\cap$ verifying $s \to^k (\alpha(l), q)$ for every $k \leq n$ then $l\sigma_i \to^\star q$ in $\mathcal{A}$.*
- *Consider now $\{\tau_i\}_{i \in I}$ defined by a term s such that $s \to^{n+1} (\alpha(l), q)$. Clearly s is a term on the form $f(s_1, \ldots, s_n)$, where $s_j = (l_j, q_j)$ is associated to $\{\tau_i\}_{i \in I}$ such that $\tau_i = \sigma_{1,i_1} \circ \ldots \circ \sigma_{n,i_n}$ where σ_{j,i_j} range over the set of substitutions defined by the term s_j.*

  $f(s_1, \ldots, s_n) \to_{\mathcal{A}_\cap} (\alpha(l), q)$ then l is of the form $f(l_1, \ldots, l_n)$ and there exists some states q_j verifying $s_j \to_{\mathcal{A}_\cap} (\alpha(l_j), q_j)$. We use the inductive hypothesis

$s_j \to^k (\alpha(l_j, q_j)$ with $k \leq n$, then $l_j\sigma_j \to^\star q_j$ in $\mathcal{A}$ and for all τ_i there exists a combination of $\sigma_{i,j}$ verifying $l\tau_i \to f(l_1\sigma_{1,i_1}, \ldots, l_n\sigma_{n,i_n}) \to f(q_1, \ldots, q_n) \to^\star_\Delta q$

2. *We use now a structural induction over l to prove that if $l\sigma \to^\star_\Delta q$ then σ correspond to a term s in $\mathcal{L}(\mathcal{A}_\cap)$:*

   (a) *if l is a constant a*
   On one hand $a \to_{\mathcal{A}_t} \alpha(a)$ and on an other hand there exists a state q in $\mathcal{Q}$ such that $a \to_\mathcal{A} q$ then $(a, a) \to_{\mathcal{A}_\cap} (\alpha(a), q)$ and $(\alpha(a), q)$ is a final state of $\mathcal{A}_\cap$, $\mathcal{A}_\cap$ recognized (a, a), that defines the empty substitution.

   (b) *If l is a variable x*
   $x \to_{\mathcal{A}_t} \alpha(x)$ and all the terms recognized by $\mathcal{A}$ may be an instance of x, we have $\{(x, q) \to (\alpha(x), q) | q \in \mathcal{Q}\} \subset \Delta_\cap$ and $\mathcal{Q}_{f,\cap} = \{\alpha(x)\} \times \mathcal{Q}$ we have defined all the $\mathcal{Q}$-substitution $\{\{x \mapsto q\} | q \in \mathcal{Q}\}$.

   (c) *If l is on the form $f(l_1, \ldots, l_n)$*
   $\mathcal{A}$ recognized $l\sigma$ then there exist some states $q, q_1, \ldots q_n$ in $\mathcal{A}$ such that $f(q_1, \ldots, q_n) \to^\star_\mathcal{A} q$, $l_i\sigma \to_\mathcal{A} q_i$. Assume that $s = f((\alpha(l_1), q_1), \ldots, (\alpha(l_n), q_n))$, there exists q_i in $\mathcal{A}$ and q_i recognized $t_i\sigma$ then thanks to induction hypothesis there are some terms s_i in $\mathcal{A}_\cap$ that define some substitutions σ_i verifying $\sigma = \sigma_1 \circ \ldots \sigma_n$. If $\sigma_1 \circ \ldots \sigma_n$ is not defined then $l\sigma$ is not recognized by $\mathcal{A}$, contradiction ; else $\sigma_1 \circ \ldots \sigma_n$ exists and $\mathcal{A}_\cap$ recognized $s = f(s_1, \ldots, s_n)$ and s define σ.

7.3 Matching in practice

Using the optimized matching algorithm in Timbuk divided by 6 the computation times for completion without increasing memory usage. In this part, we want to give an idea of the efficiency of the completion procedure with an optimized matching algorithm. Since as far as we know, there exists no other implementation of a similar algorithm, we chose to compare with a rewriting tool. Elan. Elan [2] is a very fast implementation of rewriting where rewrite rules are directly compiled into C code. Given a term rewriting system $\mathcal{R}$, we chose to use Elan and Timbuk for proving $s \to_\mathcal{R}{}^\star t$ by breadth-first search. The breadth-first search strategy for Elan was given by P.-E. Moreau. For using Timbuk to prove reachability starting on a single term s, we use the exact case (and Corollary 1) and automatic abstraction function construction.

Example 12 *Let $\mathcal{R}$ be the following TRS:*

$plus(O, x) \to x$
$plus(s(x), y) \to s(plus(x, y))$
$mult(O, x) \to x$
$mult(s(x), y) \to plus(y, mult(x, y))$
$fact(O) \to s(O)$
$fact(s(x)) \to mult(s(x), fact(x))$

To check if $fact(s^4(O)) \to_{\mathcal{R}}^* s^{24}(O)$ *Elan takes only some milliseconds where* Timbuk *takes more than* 6 *seconds. Similarly, to check if* $fact(s^5(O)) \to_{\mathcal{R}}^* s^120(O)$ *Elan takes less than a second and* Timbuk *takes more than* 11 *minutes.*

The above example and computation time are clearly not in favor of Timbuk. However, the above TRS is terminating and confluent and thus the rewriting tree is narrow. Now let us consider another example where $\mathcal{R}$ is neither terminating nor confluent TRS and the rewriting tree is wider.

Example 13 *Let* $\mathcal{R}$ *be the following TRS:*
$f(x) \to f(s(x))$
$s(s(x)) \to f(x)$
$f(s(x)) \to f(f(x))$

To check if $f(a) \to_{\mathcal{R}}^* f^6(a)$ *Elan takes more than* 18 *seconds where* Timbuk *takes only* 1 *second for the same task. Similarly, to check if* $f(a) \to_{\mathcal{R}}^* f^8(a)$ *Elan takes more than* 10 *minutes, where* Timbuk *takes only* 2 *seconds.*

Timbuk is clearly not as fast as Elan for rewriting but Timbuk takes advantage of tree automata structure (which provide some kind of sharing). Thus, for achieving reachability testing, when finite sets of very similar terms are rewritten, and when systems are neither confluent nor terminating, Timbuk obtains some results close or even better than those of Elan. This shows that for, reachability testing over non-terminating or non confluent term rewriting systems, the data structure used to represent sets of terms and the related matching algorithm over this structure plays a central role in the efficiency of the search algorithm. Note that the above well represent the kind of TRSs that can be encountered when using reachability testing on cryptographic protocols, for instance, where rewrite rules are highly non terminating (see section 6.3).

8 Extensions

In this section, we propose an extension of the completion algorithm for dealing with conditional term rewriting systems (CTRS for short). A natural way to compute the set of reachable terms for CTRSs is to encode CTRSs into TRS and use the tree automata completion algorithm for TRS. However, as shown in [10], a completion algorithm adapted to

the specific case of CTRS is likely to give some better results in practice. This algorithm specific to the conditional case is described in this section.

A conditional term rewriting system (CTRS) over a set of ground terms $\mathcal{T}(\mathcal{F})$ is a set $\mathcal{R}$ of conditional rules $(r)t_l \to t_r$ if *cond*, where $t_l, t_r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$ and *cond* designates a conjunction of conditions that must be checked before rewriting. In this paper, conditions are pairs of terms denoted by $c_1 \downarrow c_2$ where $c_1, c_2 \in \mathcal{T}(\mathcal{F}, \mathcal{X}), (Var(c_1) \cup Var(c_2)) \subseteq Var(t_l)$; these are join conditions. Such a condition is said to be true for a substitution σ if there exists a term $u \in \mathcal{T}(\mathcal{F})$ such that $c_1\sigma$ and $c_2\sigma$ can be both rewritten by the CTRS $\mathcal{R}$ into u in a finite number of steps. Then, the rule $t_l \to t_r$ can be applied to the term $t \in \mathcal{T}(\mathcal{F})$ at position p as for a TRS. $\to_{\mathcal{R}}$ also defines a rewriting relation on $\mathcal{T}(\mathcal{F})$ and thus the set of reachable terms $\mathcal{R}^\star(E)$ is defined as in the non-conditional case.

For recognizing conditions in the tree automata and compute separately their value, we will need separate states as well as the following lemma ensuring that completion builds automata where every states (not only final ones) are closed by rewriting.

Lemma 4 *[16] Let $\mathcal{R}$ be a left-linear TRS, $\mathcal{A}'$ be the result of computation of the completion algorithm applied to a set $E = \mathcal{L}(\mathcal{A})$, then $\mathcal{A}'$ is closed by rewriting w.r.t $\mathcal{R}$, i.e: if $l \to^*_{\mathcal{R}} r$ and $\exists q \in \mathcal{Q}_{\mathcal{A}}, l \in \mathcal{L}(\mathcal{A}, q)$ then $r \in \mathcal{L}(\mathcal{A}', q)$.*

8.1 Completion over regular set of terms for a CTRS

We first define a rewriting relation $t \xrightarrow{\downarrow n}_{\mathcal{R}} s$ meaning that to rewrite t into s, it is necessary to evaluate at most n recursive conditions (n is called the depth of the derivation in [9]).

Definition 18 *For a CTRS $\mathcal{R}$ with a subset $\mathcal{R}_{nc}$ of non conditional rules, we note $\xrightarrow{\downarrow n}_{\mathcal{R}}$ the relation defined by:*

- $\xrightarrow{\downarrow 0}_{\mathcal{R}} = \to_{\mathcal{R}_{nc}}$
- $a \xrightarrow{\downarrow n+1}_{\mathcal{R}} b \Leftrightarrow a \xrightarrow{\downarrow 0}_{\mathcal{R}} b$ *or* $\exists \sigma$ *substitution*, $p \in \mathcal{P}os(a)$ *and* $(l \to r \text{ if } s \downarrow t) \in \mathcal{R}$ *such that* $a|_p = l\sigma, b = a[r\sigma]_p$ *and* $\exists u \in \mathcal{T}(\mathcal{F})$ *such that* $s\sigma \xrightarrow{\downarrow n}{}^*_{\mathcal{R}} u$ *and* $t\sigma \xrightarrow{\downarrow n}{}^*_{\mathcal{R}} u$.

Note that $l \to^*_{\mathcal{R}} r$ means that $\exists n \in \mathbb{N}$ s.t. $l \xrightarrow{\downarrow n}{}^*_{\mathcal{R}} r$.

Let $\mathcal{A}_0$ be the tree automaton whose language E is the entry set of terms for the left-linear CTRS R. Let us consider the following algorithm, where we complete at each step the automaton $\mathcal{A}_i = \langle \mathcal{F}, \mathcal{Q}_i, \mathcal{Q}_f, \Delta_i \rangle$ to an automaton $\mathcal{A}_{i+1}$. The set of state Q_i is partitioned into three set of states: $Q_0 \cup Q_{i,new} \cup Q_{i,cond}$. Q_0 is the set of states of $\mathcal{A}_0$, $Q_{i,new}$ is a set of states produced by transition normalization and indexed by naturals, $Q_{i,cond}$ is a set of conditional states indexed by terms of $\mathcal{T}(\mathcal{F}, Q_i)$. Let α be an abstraction function. We use the following algorithm :

1. from $\mathcal{A}_i = \langle \mathcal{F}, \mathcal{Q}_i, \mathcal{Q}_f, \Delta_i \rangle$, the i^{th} step of completion, we compute the automaton $\mathcal{A}_{i+1} = \langle \mathcal{F}, \mathcal{Q}_{i+1}, \mathcal{Q}_f, \Delta_{i+1} \rangle$ with the initialization: $\mathcal{Q}_{i+1} = \mathcal{Q}_i, \Delta_{i+1} = \Delta_i$.

2. Let us consider each *critical pair* without considering the condition of the rule. A pair (q, r) of $\mathcal{Q} \times \mathcal{T}(\mathcal{F})$ is said to be critical for a rule (α) either non conditional $l \to r$, or conditional $l \to r$ if $c_1 \downarrow c_2$ where $\sigma \in \Sigma(\mathcal{Q}, \mathcal{X})$ is a regular language substitution $\sigma = \{x_1 \to q_{i_1}, x_2 \to q_{i_2}, \ldots, x_n \to q_{i_n}\}$, where $\{x_1, x_2, \ldots, x_n\} = var(l)$, if $l\sigma \to^*_{\Delta_i} q$ and $r\sigma \not\to^*_{\Delta_i} q$.

3. for all of these critical pairs, (α) is either:

   - a conditional rule: $l \to r$ if $c_1 \downarrow c_2$. There are two possibilities
     - there are no state indexed by $c_1\sigma$ or $c_2\sigma$ in the conditional subset of states of $\mathcal{Q}_i$ ($q_{c_1\sigma} \notin \mathcal{Q}_{i,cond}$ or $q_{c_2\sigma} \notin \mathcal{Q}_{i,cond}$), then we create these two states (or the one missing) and we add to the automaton $\mathcal{A}_{i+1}$ the following transitions :

       $$Norm_\alpha(c_1\sigma \to q_{c_1\sigma}) \cup Norm_\alpha(c_2\sigma \to q_{c_2\sigma})$$

     - there exists two states $q_{c_1\sigma}$ and $q_{c_2\sigma}$ in $\mathcal{Q}_i$. We have to calculate $\mathcal{L}(\mathcal{A}_i, q_{c_1\sigma}) \cap \mathcal{L}(\mathcal{A}_i, q_{c_2\sigma})$. If this set is empty, the condition is, for this completion step, considered as false. If it is not empty, then the condition is true and we go on processing the critical pair as if the rule were not conditional.
   - a non conditional one (or it is conditional and the condition has been found true in the previous step), then we add to the automaton the transitions $Norm_\alpha(r\sigma \to q)$.

4. the new automaton $\mathcal{A}_{i+1} = \langle \mathcal{F}, \mathcal{Q}_{i+1}, \mathcal{Q}_{f,i+1}, \Delta_{i+1} \rangle$ is the result of one step of completion of $\mathcal{A}_i$.

If there exists $i \in \mathbb{N}$ such that $\mathcal{A}_i = \mathcal{A}_{i+1}$, then $\mathcal{A}_i$ is the result. Remember that each time we add a transition to the automaton, we have to normalize it with new states (indexed by naturals and added in $Q_{i,new}$) and then the opportunity to make an approximation in order to limit the number of new states created for the normalization. As in the non conditional case, this completion may not have a fixed point: we may produce infinitely many new states. However, approximation techniques similar to those of section 5 apply: let Q_{cond} be the set of new states $q_{c_1\sigma}$ and $q_{c_1\sigma}$ produced by conditions, $\mathcal{Q}_{new}$ the set of new states used to normalize the transitions, one may restrict in any way the set $\mathcal{Q}_{new}$ to force completion to terminate. Note that there is no need to limit the number of states of Q_{cond}, since the number of possible conditions c_1, c_2 is finite and the number of possible σ is finite if $\mathcal{Q}_{new}$ is.

Theorem 7 *Let A_0 be a tree automaton such that $\mathcal{L}(\mathcal{A}_0) \supseteq E$ and $\mathcal{R}$ a left linear CTRS. If $\mathcal{A}'$ is the result of the completion of $\mathcal{A}_0$ w.r.t $\mathcal{R}$, then $\mathcal{L}(\mathcal{A}')$ is closed with respect to $\mathcal{R}$ and $\mathcal{R}^*(E) \subseteq \mathcal{R}^*(\mathcal{L}(\mathcal{A}_0)) \subseteq \mathcal{L}(\mathcal{A}')$*

Proof 14 *Let $\mathcal{A}' = \langle \mathcal{F}, \mathcal{Q}', \mathcal{Q}_f, \Delta' \rangle$. We prove that $\forall t \in \mathcal{T}(\mathcal{F})$ s.t. $\exists q \in \mathcal{Q}', t \in \mathcal{L}(\mathcal{A}', q)$, $\forall u \in \mathcal{T}(\mathcal{F})$ s.t. $t \to^*_{\mathcal{R}} u$, we have $u \in \mathcal{L}(\mathcal{A}', q)$. We prove by induction that $\forall n \in \mathbb{N}, q \in \mathcal{Q}', t \in \mathcal{L}(\mathcal{A}', q)$, u s.t. $t \xrightarrow{\downarrow n}{}^*_{\mathcal{R}} u$, then $u \in \mathcal{L}(\mathcal{A}', q)$*

- *If $q \in \mathcal{Q}', t \in \mathcal{L}(\mathcal{A}', q)$, and $t \xrightarrow{\downarrow 0}{}^*_{\mathcal{R}} u$ then we trivially have $u \in \mathcal{L}(\mathcal{A}', q)$. Indeed, $\xrightarrow{\downarrow 0}{}^*_{\mathcal{R}}$ means that we consider the subset of non conditional rules of $\mathcal{R}$ and then the proof follows from lemma 4.*

- *now suppose that for a given n: $\forall k \leq n, t \xrightarrow{\downarrow k}{}^*_{\mathcal{R}} u$ and $t \to^*_{\Delta'} q \Rightarrow u \to^*_{\Delta'} q$. We want to show that:*

$$t \xrightarrow{\downarrow n+1}{}^*_{\mathcal{R}} u \text{ and } t \to^*_{\Delta'} q \Rightarrow u \to^*_{\Delta'} q$$

$t \xrightarrow{\downarrow n+1}{}^*_{\mathcal{R}} u$ means that exists $\{t_1, t_2, \ldots, t_j\} \subseteq \mathcal{T}(\mathcal{F})$ such that

$$t_0 = t \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_1 \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_2 \xrightarrow{\downarrow n+1}_{\mathcal{R}} \ldots \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_{j-1} \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_j = u$$

Now we show that for every t_i, if $t_i \to^*_{\Delta'} q$ then $t_{i+1} \to^*_{\Delta'} q$, this leads to two cases:

  - *$t_i \xrightarrow{\downarrow n}_{\mathcal{R}} t_{i+1}$, then using the induction hypothesis, $t_{i+1} \to^*_{\Delta'} q$.*
  - *$t_i \not\xrightarrow{\downarrow n}_{\mathcal{R}} t_{i+1}$ and $t_i \xrightarrow{\downarrow n+1}_{\mathcal{R}} t_{i+1}$, so there exists a rule $(k)\, l \to r$ if $c_1 \downarrow c_2 \in \mathcal{R}$ a closed context $C[]$, and a substitution σ such that:*

$$t_i = C[l\sigma] \to_{\mathcal{R}} C[r\sigma] = t_{i+1} \text{ if } c_1\sigma \downarrow c_2\sigma$$
$$\text{and } \exists c \text{ s.t. } c_1\sigma \xrightarrow{\downarrow n}{}^*_{\mathcal{R}} c, \text{ and } c_2\sigma \xrightarrow{\downarrow n}{}^*_{\mathcal{R}} c$$

Since no critical pair between $\mathcal{R}$ and Δ' exists, the automaton is a fixed point for the completion and we necessarily have that $\exists q_{c_1\sigma}, q_{c_2\sigma} \in \mathcal{Q}'$. Thus, we have:

$$\begin{array}{llll} c_1\sigma & \xrightarrow{\downarrow n}{}^*_{\mathcal{R}} c & \text{and } c_1 & \to_{\Delta'} q_{c_1\sigma} \\ c_2\sigma & \xrightarrow{\downarrow n}{}^*_{\mathcal{R}} c & \text{and } c_1 & \to_{\Delta'} q_{c_2\sigma} \end{array}$$

The induction hypothesis leads to $c \in \mathcal{L}(\mathcal{A}', q_{c_1\sigma})$ and $c \in \mathcal{L}(\mathcal{A}', q_{c_2\sigma})$. Consequently, since $t_i \to^*_{\Delta'} q$, since the condition $\mathcal{L}(\mathcal{A}', q_{c_1,\sigma}) \cap \mathcal{L}(\mathcal{A}', q_{c_2,\sigma}) \neq \emptyset$ is true, and since $\mathcal{A}$ is a fixed point for the completion for automaton $\mathcal{A}$, we necessarily have $t_{i+1} \to^*_{\Delta'} q$.

We have $t = t_0 \in \mathcal{L}(\mathcal{A}', q)$, so by induction $\forall i \leq n, t_i \in \mathcal{L}(\mathcal{A}', q)$, in particular $u = t_j$.

We get the result that $t \xrightarrow{\downarrow n+1}{}^*_{\mathcal{R}} u$ and $t \to^*_{\Delta} q$ implies $u \to^*_{\Delta} q$
So $\forall n \in \mathbb{N}, t \xrightarrow{\downarrow n}{}^*_{\mathcal{R}} u$ and $t \to^*_{\Delta'} q$ implies $u \to^*_{\Delta'} q$, then $t \to^*_{\mathcal{R}} u$ and $t \to^*_{\Delta'} q$ implies $u \to^*_{\Delta'} q$. This leads us to $\forall q \in \mathcal{Q}', \mathcal{L}(\mathcal{A}', q)$ is closed under rewriting by $\mathcal{R}$, in particular for $q \in \mathcal{Q}_f$, thus $\mathcal{L}(\mathcal{A}')$ is closed under rewriting by $\mathcal{R}$. Since completion is incremental, we have the inequalities $\Delta \subseteq \Delta'$ and thus $E \subseteq \mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{A}')$, and finally $\mathcal{R}^*(E) \subseteq \mathcal{L}(\mathcal{A}')$.

9 Conclusion

In this paper, we have presented some tools for dealing in practice with the reachability and the unreachability problem, i.e. given two terms s and t and a term rewriting system $\mathcal{R}$ show that $s \rightarrow_{\mathcal{R}}^{\star} t$ or on the opposite show that $s \not\rightarrow_{\mathcal{R}}^{\star} t$.

The proposed algorithm, called tree automata completion, constructs a tree automaton recognizing $\mathcal{R}^{\star}(E)$: the set of terms reachable by rewriting terms of the initial regular set E with a TRS $\mathcal{R}$. The proposed algorithm is parametrized by an abstraction function and can be adapted to several purposes.

By choosing an injective abstraction function, exactness of the algorithm is guaranteed if it terminates. This result provides a new decidable class that includes (and is strictly greater) to all the decidable classes of the literature. A first result is that we thus have an alternative proof of regularity and an alternative algorithm for the known decidable cases. In some cases, like for the constructor system case, the resulting algorithm seems to be simpler to the initial algorithm.

A second result is that all those regular classes can uniformly be implemented by a single uniform algorithm that covers all of them at the same time. As far as we know this is the only implementation for those decidable classes.

Thirdly, outside of those decidable classes when completion does not terminate, using the abstraction function as an approximation tool permits to force completion to terminate on an automaton recognizing an over-approximation of $\mathcal{R}^{\star}(E)$.

To sum up, the same completion algorithm is able to build exactly $\mathcal{R}^{\star}(E)$ when it is possible and build an over-approximation otherwise. Those techniques have been implemented in the Timbuk tool which thus permits to compute $\mathcal{R}^{\star}(E)$ in the decidable classes as well as an over-approximation otherwise. Using this prototype on practical examples has shown that efficiency of the tree automata completion algorithm strongly depend on the efficiency of the matching of left-hand side of rules on tree automata. So, we proposed an optimised algorithm for matching making it possible to Timbuk to handle completion on large TRSs or large tree automata. We also showed that resulting performances makes Timbuk usable and even more relevant than usual rewriting tools to check reachability even on finite sets of terms when dealing with non-terminating TRSs. This may be of interest for proof search in theorem provers or proof assistant when the used equational theories cannot be oriented into confluent and terminating TRSs.

Note also that since approximations are only sets of first order terms, it is also possible to use approximations to perform abstract interpretation over the theories manipulated by a proof assistant and make proof more automatic. This is what is done in [25] for proving automatically some lemmas in Isabelle/HOL [24] by approximation.

The construction of a tree automaton recognizing exactly or not the set of reachable terms turns out to have several practical applications: reachability testing, sufficient completeness, strong non-termination proofs, etc. Among all those applications, reachability testing has been successfully used for cryptographic protocol verification on some real cases.

Finally, the tree automata completion algorithm can be extended to tackle the problem of approximating reachable terms for any join conditional term rewriting system. As far as we

know, this is the first time that this problem is addressed. This extension is rather natural w.r.t. the existing algorithm and uses similar techniques, in particular for approximation construction. These results suggest that some syntactic classes of CTRSs having regular sets of descendants can certainly be defined by imposing the same syntactic constraints on the right-hand side of the rules for TRS than on every right-hand side of rules and left and right-hand side of every condition for CTRS. Like in the non conditional case, those regular classes are likely to be built using the tree automata completion algorithm for the conditional case.

References

[1] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[2] P. Borovanský, C. Kirchner, H. Kirchner, P.-E. Moreau, and C. Ringeissen. An overview of elan. In *Proc. 2nd WRLA*, ENTCS, Pont-à-mousson (France), 1998. Elsevier.

[3] Walter S. Brainerd. Tree generating regular systems. *Information and Control*, 14:217–231, 1969.

[4] H. Comon. Sufficient completeness, term rewriting system and anti-unification. In J. Siekmann, editor, *Proc. 8th CADE Conf., Oxford (UK)*, volume 230 of *LNCS*, pages 128–140. Springer-Verlag, 1986.

[5] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. `http://www.grappa.univ-lille3.fr/tata/`, 2002.

[6] J.L. Coquidé, M. Dauchet, R. Gilleron, and S. Vágvölgyi. Bottom-up tree pushdown automata and rewrite systems. In R. V. Book, editor, *Proc. 4th RTA Conf., Como (Italy)*, volume 488 of *LNCS*, pages 287–298. Springer-Verlag, 1991.

[7] M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *Proc. 5th LICS Symp., Philadelphia (Pa., USA)*, pages 242–248, June 1990.

[8] N. Dershowitz and J.-P. Jouannaud. *Handbook of Theoretical Computer Science*, volume B, chapter 6: Rewrite Systems, pages 244–320. Elsevier Science Publishers B. V. (North-Holland), 1990. Also as: Research report 478, LRI.

[9] N. Dershowitz, M. Okada, and G. Sivakumar. Canonical conditional rewrite systems. In *Proc. 9th CADE Conf., Argonne (Ill., USA)*, volume 310 of *LNCS*. Springer-Verlag, May 1988.

[10] G. Feuillade and T. Genet. Reachability in conditional term rewriting systems. In *FTP'2003, International Workshop on First-Order Theorem Proving*, volume 86 n. 1 of *ENTCS*. Elsevier, June 2003.

[11] T. Genet. Decidable approximations of sets of descendants and sets of normal forms. In *Proc. 9th RTA Conf., Tsukuba (Japan)*, volume 1379 of *LNCS*, pages 151–165. Springer-Verlag, 1998.

[12] T. Genet and F. Klay. Rewriting for Cryptographic Protocol Verification. In *Proc. 17th CADE Conf., Pittsburgh (Pen., USA)*, volume 1831 of *LNAI*. Springer-Verlag, 2000.

[13] T. Genet, Y.-M. Tang-Talpin, and V. Viet Triem Tong. Verification of Copy Protection Cryptographic Protocol using Approximations of Term Rewriting Systems. Workshop on Issues in the Theory of Security, 2003.

[14] T. Genet and V. Viet Triem Tong. Timbuk Documentation. IRISA / Université de Rennes 1, 2001. `http://www.irisa.fr/lande/genet/timbuk/`.

[15] T. Genet and Valérie Viet Triem Tong. Reachability Analysis of Term Rewriting Systems with *timbuk*. In *Proc. 8th LPAR Conf., Havana (Cuba)*, volume 2250 of *LNAI*, pages 691–702. Springer-Verlag, 2001.

[16] Thomas Genet. Decidable approximations of sets of descendants and sets of normal forms (extended version). Technical Report RR-3325, INRIA, 1997.

[17] R. Gilleron and S. Tison. Regular tree languages and rewrite systems. *Fundamenta Informaticae*, 24:157–175, 1995.

[18] F. Jacquemard. Decidable approximations of term rewriting systems. In H. Ganzinger, editor, *Proc. 7th RTA Conf., New Brunswick (New Jersey, USA)*, pages 362–376. Springer-Verlag, 1996.

[19] D. Kapur, P. Narendran, and H. Zhang. On sufficient completeness and related properties of term rewriting systems. *Acta Informatica*, 24:395–415, 1987.

[20] Donald E. Knuth and P. B. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.

[21] E. Kounalis. Completeness in data type specifications. In B. Buchberger, editor, *Proceedings EUROCAL Conference, Linz (Austria)*, volume 204 of *LNCS*, pages 348–362. Springer-Verlag, 1985.

[22] X. Leroy, D. Doligez, J. Garrigue, D. Rémy, and J. Vouillon. The Objective Caml system release 3.00 – Documentation and user's manual. INRIA, 2000. `http://caml.inria.fr/ocaml/htmlman/`.

[23] T. Nipkow and G. Weikum. A decidability result about sufficient completeness of axiomatically specified abstract data types. In *6th GI Conference*, volume 145 of *LNCS*, pages 257–268. Springer-Verlag, 1983.

[24] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[25] F. Oehl and D. Sinclair. Combining two approaches for the formal verification of cryptographic protocols. In *Proceedings of ICLP Workshop on Specification, Analysis and Validation for Emerging technologies in computational logic*, 2001.

[26] L. Paulson. Proving Properties of Security Protocols by Induction. In *10th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1997.

[27] P. Réty. Regular Sets of Descendants for Constructor-based Rewrite Systems. In *Proc. 6th LPAR Conf., Tbilisi (Georgia)*, volume 1705 of *LNAI*. Springer-Verlag, 1999.

[28] K. Salomaa. Deterministic Tree Pushdown Automata and Monadic Tree Rewriting Systems. *J. of Computer and System Sciences*, 37:367–394, 1988.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399